

10715398

## PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2001-157145

(43)Date of publication of application : 08.06.2001

(51)Int.Cl.

H04N 5/76  
G11B 20/10  
G11B 27/034  
H04N 5/85  
H04N 5/93

(21)Application number : 11-332352

(71)Applicant : SONY CORP

(22)Date of filing : 24.11.1999

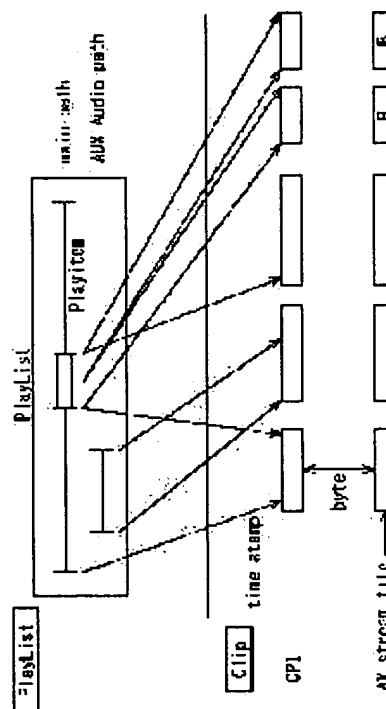
(72)Inventor : HAMADA TOSHIYA  
KATO MOTOKI

## (54) RECORDING AND REPRODUCING DEVICE AND METHOD, AND RECORDING MEDIUM

(57)Abstract:

PROBLEM TO BE SOLVED: To reproduce an AV signal without seams.

SOLUTION: Information showing the state (A type, C type, D type or E type) of an IN point and an OUT point on a Clip designated by a Playitem is described in a Playitem, concerning a Playlist where at least one and more Playitems are arranged in order of reproduction.



(19)日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11)特許出願公開番号

特開2000-194843

(P2000-194843A)

(43)公開日 平成12年7月14日(2000.7.14)

(51)IntCl. <sup>7</sup>	識別記号	F I	テマコード* (参考)
G 0 6 T 3/60		G 0 6 F 15/66	3 5 0 A
H 0 3 M 7/30		H 0 3 M 7/30	Z
H 0 4 N 1/41		H 0 4 N 1/41	Z
	7/24		A

審査請求 未請求 請求項の数83 OL (全 43 頁)

(21)出願番号	特願平11-342646	(71)出願人	390019839 三星電子株式会社 大韓民国京畿道水原市八達区梅灘洞416
(22)出願日	平成11年12月1日(1999.12.1)	(72)発明者	金 成珍 大韓民国京畿道水原市八達区梅灘4洞810番地 三星1次アパート2棟406号
(31)優先権主張番号	1998P52327	(72)発明者	榮 文燮 大韓民国京畿道龍仁市器興邑農書里山14-1番地
(32)優先日	平成10年12月1日(1998.12.1)	(74)代理人	100094145 弁理士 小野 由己男 (外1名)
(33)優先権主張国	韓国 (KR)		
(31)優先権主張番号	1999P3649		
(32)優先日	平成11年2月4日(1999.2.4)		
(33)優先権主張国	韓国 (KR)		
(31)優先権主張番号	1999P5239		
(32)優先日	平成11年2月13日(1999.2.13)		
(33)優先権主張国	韓国 (KR)		

最終頁に続く

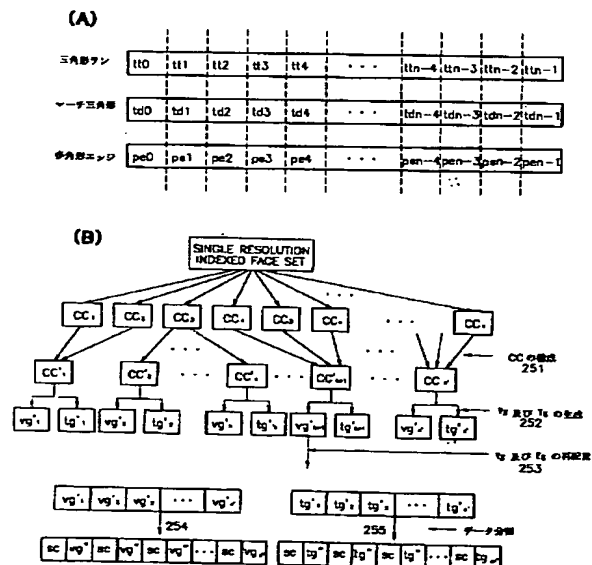
(54) 【発明の名称】 エラー回復性を有する漸進的三次元メッシュ情報の符号化／復号化方法

(57) 【要約】

【課題】 MPEG-4合成／天然混合符号化分野及び仮想現実モデリング言語で使われている三次元メッシュデータをエラー回復的、漸進的に復元する特性を有するように符号化／復号化する方法を提供する。

【解決手段】 多角形三次元メッシュの漸進的な復元を可能にする符号化方法において、（a） 多角形三次元メッシュを1つ以上の連結成分に分割する段階と、

(b) 各連結成分に対して頂点グラフ情報及び三角形ツリー／三角形データ情報を生成する段階と、(c) 各連結成分に対し、その連結成分を構成する頂点グラフ情報及び三角形ツリー／三角形データ情報を、独立的に復号化可能な形式を有する基本メッシュの形式に合せて符号化する段階とを含み、三次元メッシュデータの漸進的な符号化及びデータ損失に対する回復性が与えられるように連結情報、幾何情報及び画像情報からなる三次元メッシュデータを符号化／復号化する方法である。



1

## 【特許請求の範囲】

【請求項 1】 多角形三次元メッシュの漸進的な復元を可能にする符号化方法において、

(a) 多角形三次元メッシュを 1 つ以上の連結成分に分割する段階と、

(b) 各連結成分に対して頂点グラフ情報及び三角形ツリー/三角形データ情報を生成する段階と、

(c) 各連結成分に対し、その連結成分を構成する頂点グラフ情報及び三角形ツリー/三角形データ情報を、独立10的に復号化可能な形式を有する基本メッシュの形式に合せて符号化する段階と、を含むことを特徴とする多角形三次元メッシュ情報の符号化方法。

【請求項 2】 前記(a)段階は、

多角形三次元メッシュを多数の段階別メッシュに分類し、各段階別メッシュに分類された多角形三次元メッシュを 1 つ以上の連結成分に分割する段階であることを特徴とする請求項 1 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 3】 前記(c)段階において、

一つの連結成分を構成する三角形ツリー/三角形データ情報が基本メッシュの形式に合わせられない程度に大きな場合には、前記三角形ツリー/三角形データ情報を独立して復号化可能な形式を有するデータパーティションに分割して基本メッシュの形式に合せて符号化し、多数の連結成分を構成する情報が基本メッシュの形式に合わせられる場合には、多数の連結成分を構成する情報を一つのデータパーティションにして基本メッシュの形式に合せて符号化することを特徴とする請求項 1 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 4】 前記三角形ツリー/三角形データ情報が符20号化された基本メッシュは、

三角形ツリーのメインブランチに位置した分岐三角形が存在する所に 1 ビットの方向性表示子を具備し、その方向性表示子に表示された方向に応じて前記分岐三角形に繋がる従属ツリーの符号化順序が決定されることを特徴とする請求項 1 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 5】 前記方向性表示子は、

前記分岐三角形とつながる従属ツリーの大きさ情報に応じて決定されることを特徴とする請求項 4 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 6】 前記従属ツリー内における三角形の訪問順序は、

全ての従属ツリーに対して同一な方向であることを特徴とする請求項 5 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 7】 前記従属ツリー内における三角形の訪問順序は、

前記方向性表示子の値に対応して決定されることを特徴とする請求項 5 に記載の多角形三次元メッシュ情報の符

(2)

特開 2000-194843

2

号化方法。

【請求項 8】 前記三角形ツリー/三角形データ情報が符号化された基本メッシュは、

三角形ツリーのうちメインブランチに位置した分岐三角形から任意の側ブランチに位置した従属ツリーの総大きさ情報を具備し、この総大きさ情報により Y-頂点のインデックスが決定されることを特徴とする請求項 1 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 9】 多角形三次元メッシュを漸進的、エラー回復的に復元可能に符号化する方法において、

(a) 多角形三次元メッシュを 1 つ以上の連結成分に分割する段階と、

(b) 各連結成分に対して頂点グラフ情報及び三角形ツリー/三角形データ情報を生成する段階と、

(c) 各連結成分に対して、その連結成分を構成する頂点グラフ情報及び三角形ツリー/三角形データ情報を、独立して復号化可能に固定されたビットストリーム形式を有する基本メッシュの形式に合せて符号化する段階とを含むことを特徴とする多角形三次元メッシュ情報の符号化方法。

【請求項 10】 前記(c)段階は、

前記多角形三次元メッシュを構成する全ての連結成分に対して、頂点グラフ情報、三角形ツリー/三角形データ情報の順に配列して一つのデータパーティションで構成することを特徴とする請求項 9 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 11】 前記(c)段階は、

前記多角形三次元メッシュを構成する全ての連結成分に対応する頂点グラフ情報を先に配列し、全ての連結成分に対応する三角形ツリー/三角形データ情報を後で配列して一つのデータパーティションで構成して符号化することを特徴とする請求項 9 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 12】 前記(c)段階は、

各連結成分別にその連結成分を構成する情報を独立したデータパーティションで構成して符号化することを特徴とする請求項 9 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 13】 前記(c)段階は、

前記多角形メッシュを構成する全ての連結成分に対応する頂点グラフ情報を一つのデータパーティションで構成し、全ての連結成分に対応する三角形ツリー/三角形データ情報を他の一つのデータパーティションで構成して符号化することを特徴とする請求項 9 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 14】 前記(c)段階は、

前記多角形メッシュを構成する全ての連結成分に対応する頂点グラフ情報を一のデータパーティションで構成し、各連結成分に対応する三角形ツリー/三角形データ情報を各々独立したデータパーティションで構成する

3

が、伝送パケットに比べて大きな連結成分の三角形ツリー/三角形データ情報は独立して復号化可能な多数のデータパーティションに分割して符号化することを特徴とする請求項9に記載の多角形三次元メッシュ情報の符号化方法。

【請求項15】 前記(c)段階は、各連結成分を構成する頂点グラフ情報及び三角形ツリー/三角形データ情報別に各々独立したデータパーティションを構成して符号化することを特徴とする請求項9に記載の多角形三次元メッシュ情報の符号化方法。

【請求項16】 前記(c)段階は、各連結成分に対応する頂点グラフ情報を各々独立したデータパーティションで構成して先に符号化し、各連結成分に対応する三角形ツリー/三角形データ情報を各々独立したデータパーティションで構成するが、伝送パケットに比べて大きな連結成分の三角形ツリー/三角形データ情報は、独立して復号化可能な多数の分割されたデータパーティションで構成して後で符号化することを特徴とする請求項9に記載の多角形三次元メッシュ情報の符号化方法。

【請求項17】 前記(c)段階は、各連結成分に対応する頂点グラフ情報別に独立したデータパーティションで構成し、三角形ツリー/三角形データ情報別に独立したデータパーティションで構成して連結成分別に符号化することを特徴とする請求項9に記載の多角形三次元メッシュ情報の符号化方法。

【請求項18】 前記(c)段階は、各連結成分に対応する頂点グラフ情報別に独立したデータパーティションで構成し、三角形ツリー/三角形データ情報別に独立したデータパーティションで構成するが、伝送パケットに比べて大きな連結成分の三角形ツリー/三角形データ情報は、独立して復号化可能な形態に分割されたデータパーティションで構成して連結成分別に符号化することを特徴とする請求項9に記載の多角形三次元メッシュ情報の符号化方法。

【請求項19】 多角形三次元メッシュを漸進的でエラー回復的に復元可能に符号化する方法において、

(a) 多角形三次元メッシュを1つ以上の連結成分に分割する段階と、

(b) 各連結成分に対して頂点グラフ情報及び三角形ツリー/三角形データ情報を生成する段階と、

(c) 各連結成分に対して、その連結成分を構成する頂点グラフ情報及び三角形ツリー/三角形データ情報を、独立して復号化可能な情報の特性に応じて可変的なビットストリームの形式を有する基本メッシュの形式に合せて符号化する段階とを含むことを特徴とする多角形三次元メッシュ情報の符号化方法。

【請求項20】 前記(c)段階は、前記データパーティションの開始コード内の一部ビットをパーティション類型で使用する符号化されたビットス

(3)

特開2000-194843

4

トリームの構造を示すことを特徴とする請求項19に記載の多角形三次元メッシュ情報の符号化方法。

【請求項21】 前記パーティション類型には、1つまたは多数の連結成分に対応する情報を集めて一つのデータパーティションで構成して符号化する第0パーティション類型が備えられることを特徴とする請求項20に記載の多角形三次元メッシュ情報の符号化方法。

【請求項22】 前記第0パーティション類型が符号化された基本メッシュには、

10 各連結成分の符号化後、符号化されるべき連結成分がさらに存在するか否かを示すためのビットがさらに備えられることを特徴とする請求項21に記載の多角形三次元メッシュ情報の符号化方法。

【請求項23】 前記パーティション類型には、1つまたは多数の連結成分に対応する頂点グラフ情報を集めて一つのデータパーティションで構成して符号化する第1パーティション類型が備えられることを特徴とする請求項20に記載の多角形三次元メッシュ情報の符号化方法。

20 【請求項24】 前記第1パーティション類型が符号化された基本メッシュには、各連結成分に対応する頂点グラフ情報の符号化後、符号化されるべき頂点グラフ情報がさらに存在するか否かを示すためのビットがさらに備えられることを特徴とする請求項23に記載の多角形三次元メッシュ情報の符号化方法。

【請求項25】 前記パーティション類型には、三角形ツリー/三角形データ情報を分割してデータパーティションで構成して符号化する第2パーティション類型が備えられることを特徴とする請求項20に記載の多角形三次元メッシュ情報の符号化方法。

30 【請求項26】 前記第2パーティション類型が符号化された基本メッシュには、前記三角形ツリー/三角形データ情報に対応する頂点グラフの識別子がさらに備えられることを特徴とする請求項25に記載の多角形三次元メッシュ情報の符号化方法。

40 【請求項27】 前記三角形ツリー/三角形データ情報が参照しうる頂点グラフが1つだけである場合には、前記頂点グラフの識別子を符号化しないことを特徴とする請求項26に記載の多角形三次元メッシュ情報の符号化方法。

【請求項28】 前記第2パーティション類型が符号化された基本メッシュには、再生されるべき三角形ストリップのために境界ループテーブルにおける開始インデックスをさらに備えることを特徴とする請求項25に記載の多角形三次元メッシュ情報の符号化方法。

50 【請求項29】 前記第2パーティション類型が符号化された基本メッシュは、

5

二つ以上のデータパーティションにおいて共通の幾何情報と画像情報とをどのように予測するかを示す境界予測表示子をさらに備えることを特徴とする請求項25に記載の多角形三次元メッシュ情報の符号化方法。

【請求項30】 前記パーティション類型は、一つの連結成分を一つのデータパーティションで構成して符号化する第3パーティション類型を備えることを特徴とする請求項20に記載の多角形三次元メッシュ情報の符号化方法。

【請求項31】 三角形三次元メッシュを漸進的でエラー回復的に復元可能に符号化する時、三角形三次元メッシュをデータパーティションに分割してバケット化する方法において、

(a) 三角形ツリーに含まれた三角形を順次に訪問しながらその三角形における総ビット発生量を計算する段階と、

(b) 前記(a)段階で計算された三角形の総ビット発生量を累積する段階と、

(c) 前記(b)段階で累積した値がバケット大きさにバケット許容値をかけた値より小さな場合には三角形ツリーに含まれた次の訪問三角形に対して前記(a)段階以降を繰返し、小さくない場合には前記訪問された三角形までの三角形ツリー/三角形データ情報をデータパーティションに分割してバケット化する段階と、を含むことを特徴とする三角形三次元メッシュのデータパーティションへの分割方法。

【請求項32】 前記(a)段階で訪問された三角形が分岐三角形である場合には、

(d) 前記(b)段階で累積した値に符号化順序によって決定された従属ツリーに含まれた全ての三角形を符号化したビット発生量を合せた値が、前記バケット大きさに前記バケット許容値をかけた値より小さな場合には前記(a)段階以降を繰返し、小さくない場合には前記分岐三角形までの三角形ツリー/三角形データ情報をデータパーティションに分割してバケット化する段階をさらに含むことを特徴とする請求項31に記載の三角形三次元メッシュのデータパーティションへの分割方法。

【請求項33】 前記(a)段階において訪問された三角形が分岐三角形である場合には、

(d) 前記(b)段階で累積した値に符号化順序によって決定された従属ツリーの大きさに応じて推定されるビット発生量を合せた値が、前記バケット大きさに前記バケット許容値をかけた値より小さな場合には前記(a)段階以降を繰返し、小さくない場合には前記分岐三角形までの三角形ツリー/三角形データ情報をデータパーティションに分割してバケット化する段階をさらに含むことを特徴とする請求項31に記載の三角形三次元メッシュのデータパーティションへの分割方法。

【請求項34】 多角形三次元メッシュを漸進的でエラー回復的に復元可能に符号化する方法において、

(4)

特開2000-194843

6

(a) 多角形三次元メッシュを1つ以上の連結成分に分割する段階と、

(b) 各連結成分に対して頂点グラフ情報及び三角形ツリー/三角形データ情報を生成する段階と、

(c) 各連結成分に対してその連結成分を構成する前記頂点グラフ情報を符号化する段階と、

(d) 前記三角形ツリー/三角形データ情報を分割したデータパーティションに各々仮想ビット対を追加することによって仮想連結成分で構成して符号化する段階と、を含むことを特徴とする多角形三次元メッシュ情報の符号化方法。

【請求項35】 前記(d)段階の分割は、三角形ツリーのメインランチでのみなされることを特徴とする請求項34に記載の多角形三次元メッシュ情報の符号化方法。

【請求項36】 前記(d)段階の分割が、三角形ツリーのランまたはリーフで行われる場合には(t<sub>run</sub>, t<sub>leaf</sub>)対に一对の(1, 1)を追加することにより仮想連結成分を構成することを特徴とする請求項35に記載の多角形三次元メッシュ情報の符号化方法。

【請求項37】 前記(d)段階の分割が、三角形ツリーに分岐三角形で行われる場合には(t<sub>run</sub>, t<sub>leaf</sub>)対に二対の(1, 1)を追加することによって仮想連結成分を構成することを特徴とする請求項35に記載の多角形三次元メッシュ情報の符号化方法。

【請求項38】 前記(a)段階以前に、前記多角形三次元メッシュを三角形三次元メッシュで再構成する段階をさらに含むことを特徴とする請求項34に記載の多角形三次元メッシュ情報の符号化方法。

【請求項39】 前記(d)段階の分割は、多角形メッシュの実際のエッジでのみ行われ、前記三角形ツリー/三角形データのデータパーティションに含まれた最初の三角形の多角形エッジ情報は符号化しないことを特徴とする請求項38に記載の多角形三次元メッシュ情報の符号化方法。

【請求項40】 前記(d)段階の分割は、多角形メッシュの実際のエッジ及び仮想エッジで行われ、前記三角形ツリー/三角形データの第0パーティション類型に含まれた最初の三角形の多角形エッジ情報は符号化しなく、残りパーティション類型では符号化することを特徴とする請求項38に記載の多角形三次元メッシュ情報の符号化方法。

【請求項41】 前記仮想連結成分で構成されたデータパーティション内に存在する全ての分岐三角形のうちメインランチに位置する分岐三角形に対し、1ビットの方向性表示子を含むことを特徴とする請求項34に記載の多角形三次元メッシュ情報の符号化方法。

【請求項42】 前記三角形ツリー/三角形データ情報を分割したデータパーティションのヘッダは、以前に復元されたデータパーティションとは独立して復

50

(5)

特開 2000-194843

7

元可能に境界ループテーブルにおける開始インデックスを具備することを特徴とする請求項 3 4 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 4 3】 前記開始インデックスは、三角形ツリー/三角形データ情報と対応する境界ループテーブルにおいて連続復元される三角形の開始頂点に対応する境界ループにおけるマッピングされる最初の左右インデックスに決め、境界ループのインデックスは構成成分の頂点数に 2 を足した範囲内で決めることを特徴とする請求項 4 2 に記載の三次元メッシュ情報の符号化方法。

【請求項 4 4】 前記開始インデックスは、三角形ツリー/三角形データ情報と対応する境界ループテーブルにおいて連続復元される三角形の開始頂点に対応する境界ループにおけるマッピングされる最初の左右インデックスに決め、境界ループのインデックスは全体メッシュを考慮して決めることを特徴とする請求項 4 2 に記載の三次元メッシュ情報の符号化方法。

【請求項 4 5】 前記三角形ツリー/三角形データ情報を分割したデータパーティションのヘッダには、前記三角形ツリー/三角形データ情報に対応する頂点グラフの識別子がさらに備えられることを特徴とする請求項 3 4 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 4 6】 前記三角形ツリー/三角形データ情報を分割したデータパーティションのヘッダには、二つ以上のデータパーティションにおいて共通の幾何情報と画像情報とをどのように予測するかを示す境界予測表示子が備えられることを特徴とする請求項 3 4 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 4 7】 前記データパーティション始終が三角形ツリーのメインブランチに位置する場合には、前記データパーティションと以前に符号化されたデータパーティションに共通の幾何情報及び画像情報を 1 つのデータパーティションにのみ含ませ、前記データパーティションの始終が従属ツリーに位置する場合には、前記データパーティションと以前に符号化されたデータパーティションに共通の幾何情報及び画像情報の存否を判別し、存在する場合には二つのデータパーティションに前記情報を全て含ませ、そうでない場合には 1 つのデータパーティションにのみ含ませることを特徴とする請求項 4 6 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 4 8】 三角形ツリー/三角形データ情報に含まれた三角形データ情報は、対応する三角形ツリーを構成する三角形別にマーチ情報、幾何情報及び画像情報順に配列されることを特徴とする請求項 3 4 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 4 9】 三角形ツリー/三角形データ情報に含まれた三角形データ情報は、対応する三角形ツリーを構成する全ての三角形のマーチ情報、全ての三角形の幾何情

8

報及び全ての三角形の画像情報順に配列されることを特徴とする請求項 3 4 に記載の多角形三次元メッシュ情報の符号化方法。

【請求項 5 0】 多角形三次元メッシュを漸進的でエラー回復的に復号化する方法において、

(a) 入力されたビットストリームを基本メッシュ単位に区分する段階と、

(b) 前記基本メッシュのパーティション類型を判断する段階と、

10 (c) 前記基本メッシュに頂点グラフ情報が含まれていると、前記頂点グラフ情報を復号化してバウンディングループテーブルを生成する段階と、

(d) 前記基本メッシュに三角形ツリー/三角形データ情報が含まれていると、前記三角形ツリー/三角形データ情報を復号化して三次元メッシュを生成する段階と、

(e) 前記(a)段階乃至前記(d)段階を繰返すことによって三次元メッシュを生成する段階と、を含むことを特徴とする三次元メッシュ情報の復号化方法。

【請求項 5 1】 前記(a)段階は、

20 前記入力されるビットストリームを多角形三次元メッシュ単位に区分し、多角形三次元メッシュに含まれた段階別メッシュの種類を判別して復号化可能な段階別メッシュならその段階別メッシュ内のビットストリームを基本メッシュ単位に区分することを特徴とする請求項 5 0 に記載の多角形三次元メッシュ情報の復号化方法。

【請求項 5 2】 前記(d)段階において三角形ツリー/三角形データ情報を復号化する際、

30 三角形ツリーのメインブランチに位置した分岐三角形を復号化する場合、方向性表示子に表示された方向に応じて前記分岐三角形に繋がる従属ツリーの復号化順序が決定されることを特徴とする請求項 5 0 に記載の多角形三次元メッシュ情報の復号化方法。

【請求項 5 3】 前記従属ツリー内における三角形の訪問順序は、

全ての従属ツリーに対して同一な方向であることを特徴とする請求項 5 2 に記載の多角形三次元メッシュ情報の復号化方法。

【請求項 5 4】 前記従属ツリー内における三角形の訪問順序は、

40 前記方向性表示子の値に対応して決定されることを特徴とする請求項 5 2 に記載の多角形三次元メッシュ情報の復号化方法。

【請求項 5 5】 前記(b)段階は、

前記データパーティションの開始コード値に応じてパーティション類型を判断することを特徴とする請求項 5 2 に記載の多角形三次元メッシュ情報の復号化方法。

【請求項 5 6】 前記パーティション類型には、

1 つまたは多数の連結成分に対応する情報を集めて一つのデータパーティションで構成して符号化する第 0 パーティション類型が備えられることを特徴とする請求項 5

9

5に記載の多角形三次元メッシュ情報の復号化方法。

【請求項57】 前記第0パーティション類型の基本メッシュの復号化時、

各連結成分の後に位置した所定のビットにより復号化されるべき連結成分がさらに存在するかを把握することを特徴とする請求項56に記載の多角形三次元メッシュ情報の復号化方法。

【請求項58】 前記パーティション類型には、

1つまたは多数の連結成分に対応する頂点グラフ情報を集めて一つのデータパーティションで構成して符号化する第1パーティション類型が備えられることを特徴とする請求項55に記載の多角形三次元メッシュ情報の復号化方法。

【請求項59】 前記第1パーティション類型の基本メッシュの復号化時、

各連結成分に対応する頂点グラフ情報の後に位置した所定のビットにより復号化されるべき頂点グラフ情報がさらに存在するかを把握することを特徴とする請求項58に記載の多角形三次元メッシュ情報の復号化方法。

【請求項60】 前記パーティション類型には、

三角形ツリー/三角形データ情報を分割してデータパーティションで構成して符号化する第2パーティション類型が備えられることを特徴とする請求項55に記載の多角形三次元メッシュ情報の復号化方法。

【請求項61】 前記第2パーティション類型の基本メッシュの復号化時、

前記三角形ツリー/三角形データ情報の前に位置した頂点グラフの識別子により、対応する頂点グラフ情報を識別することを特徴とする請求項60に記載の多角形三次元メッシュ情報の復号化方法。

【請求項62】 前記第2パーティション類型の基本メッシュの復号化時、

前記三角形ツリー/三角形データ情報に対応する頂点グラフ情報がユニークに確認されうる場合には、前記三角形ツリー/三角形データ情報の前に頂点グラフの識別子が位置しないと判断することを特徴とする請求項60に記載の多角形三次元メッシュ情報の復号化方法。

【請求項63】 前記第2パーティション類型の基本メッシュの復号化時、

前記三角形ツリー/三角形データ情報の前に位置した境界ループテーブルにおける開始インデックスにより以前のデータパーティションとは独立して復号化することを特徴とする請求項60に記載の多角形三次元メッシュ情報の復号化方法。

【請求項64】 前記第2パーティション類型の基本メッシュの復号化時、

前記三角形ツリー/三角形データ情報の前に位置した境界予測表示子により、以前に復号化したデータパーティションとの共通の幾何情報と画像情報とをどのように予測するかを判断することを特徴とする請求項60に記載

(6)

特開2000-194843

10

の多角形三次元メッシュ情報の復号化方法。

【請求項65】 前記パーティション類型には、

一つの連結成分を一つのデータパーティションで構成した第3パーティション類型が備えられることを特徴とする請求項55に記載の多角形三次元メッシュ情報の復号化方法。

【請求項66】 前記(d)段階において三角形ツリー/三角形データ情報の復号化時、

三角形ツリーのうちメインブランチに位置した分岐三角形を復号化する場合、任意の一方のブランチに位置した従属ツリーの総大きさ情報により前記分岐三角形におけるY-頂点インデックスを決定することを特徴とする請求項50に記載の多角形三次元メッシュ情報の復号化方法。

【請求項67】 多角形三次元メッシュを漸進的でエラー回復的に復号化する方法において、

(a) 入力されたビットストリームを基本メッシュ単位に区分する段階と、

(b) 前記基本メッシュのパーティション類型を判断する段階と、

(c) 前記基本メッシュに頂点グラフ情報が含まれていると、前記頂点グラフ情報を復号化してバウンディングループテーブルを生成する段階と、

(d) 前記基本メッシュに三角形ツリー/三角形データ情報が含まれていると、連結成分単位で前記三角形ツリー/三角形データ情報を復号化して三角形三次元メッシュを生成する段階と、

(e) 前記(d)段階における連結成分が仮想連結成分の場合前記(a)段階の以降を繰返し、そうでない場合三角形三次元メッシュの生成を完了する段階と、を含むことを特徴とする三次元メッシュ情報の復号化方法。

【請求項68】 前記(d)段階において三角形ツリー/三角形データ情報を含む基本メッシュには、

以前に復元された基本メッシュとは独立して復元可能に境界ループテーブルにおける開始インデックスが備えられることを特徴とする請求項67に記載の多角形三次元メッシュ情報の復号化方法。

【請求項69】 前記開始インデックスは、

三角形ツリー/三角形データ情報に対応する境界ループテーブルで連続復元される三角形の開始頂点に対応する境界ループにおいてマッピングされる最初の左右側インデックスとして決定され、境界ループのインデックスは構成成分の頂点数に2を足した範囲内で決定されることを特徴とする請求項68に記載の三次元メッシュ情報の復号化方法。

【請求項70】 前記開始インデックスは、

三角形ツリー/三角形データ情報に対応する境界ループテーブルで連続復元される三角形の開始頂点に対応する境界ループにおいてマッピングされる最初の左右側インデックスとして決定され、境界ループのインデックスは

10

20

30

40

50

11

全体メッシュを考慮して決定されることを特徴とする請求項68に記載の三次元メッシュ情報の復号化方法。

【請求項71】 前記(d)段階において、右側インデックスー左側インデックスー1が前記基本メッシュの三角形ツリー情報のみを復元した時発生する三角形の総数より大きな場合には、前記連結成分は仮想連結成分であると判断することを特徴とする請求項68に記載の三次元メッシュ情報の復号化方法。

【請求項72】 前記(d)段階において復号化された三角形のうち最後から3番目の三角形が分岐三角形であれば(trun、tleaf)対に二対の仮想三角形が存在すると判断し、分岐三角形でなければ(trun、tleaf)対に一对の仮想三角形が存在すると判断し、仮想三角形に対しては三角形データ情報を復号化しないことを特徴とする請求項71に記載の多角形三次元メッシュ情報の復号化方法。

【請求項73】 三角形三次元メッシュを多角形三次元メッシュで再構成する段階をさらに含むことを特徴とする請求項67に記載の多角形三次元メッシュ情報の符号化方法。

【請求項74】 符号化過程におけるパーティション分割が多角形メッシュの実際のエッジでのみ行われ、前記三角形ツリー/三角形データの基本メッシュに含まれた最初の三角形の多角形エッジは復号化無しに実際のエッジであると判断することを特徴とする請求項73に記載の多角形三次元メッシュ情報の復号化方法。

【請求項75】 符号化過程におけるパーティション分割が多角形メッシュの内部でも行われ、前記三角形ツリー/三角形データの最初の基本メッシュに含まれた最初の三角形の多角形エッジは復号化なしに実際のエッジであると判断することを特徴とする請求項73に記載の多角形三次元メッシュ情報の復号化方法。

【請求項76】 前記仮想連結成分で構成された基本メッシュ内に存在する全ての分岐三角形に対し、分岐三角形を復号化する場合、方向性表示子に表示された方向に依じて、前記分岐三角形に繋がる従属ツリーの復号化順序が決定されることを特徴とする請求項67に記載の多角形三次元メッシュ情報の復号化方法。

【請求項77】 前記(d)段階において三角形ツリー/三角形データ情報を含む基本メッシュには、二つ以上の基本メッシュにおいて共通の幾何情報と画像情報とをどのように予測するかを示す境界予測表示子が備えられることを特徴とする請求項67に記載の多角形三次元メッシュ情報の復号化方法。

【請求項78】 前記境界予測表示子が0の場合、現在復号化される基本メッシュと以前に復号化された基本メッシュに共通の全ての幾何情報及び画像情報が、現在復号化される基本メッシュには含まれていないことを示すことを特徴とする請求項77に記載の多角形三次元メッシュ情報の復号化方法。

【請求項79】 幾何情報を、現在のデータパーティシ

(7)

特開2000-194843

12

ョンにおいて以前に訪問された三つの先祖により予測した予測値と符号化された値を用いて復号化する際、必要な先祖のうち何らの値も用いられないと予測値を0とし、一つの値のみ用いられるとその値を予測値とし、二つの値のみが用いられ、その両値と現在の頂点間の距離が1ならば両値の算術平均を予測値とし、二つの値のみ用いられその両値中現在の頂点からの距離が1の値が1つであればその値を予測値とし、三つの値を全て用いられると所定の予測方法により予測値を決定することを特徴とする請求項78に記載の多角形三次元メッシュ情報の復号化方法。

【請求項80】 前記境界予測表示子が1の場合、現在復号化される基本メッシュと以前に復号化された基本メッシュに共通の全ての幾何情報及び画像情報が現在復号化される基本メッシュにも含まれていることを示すことを特徴とする請求項77に記載の多角形三次元メッシュ情報の復号化方法。

【請求項81】 幾何情報を訪問された先祖により予測した予測値と符号化された値を用いて復号化する時、以前のデータパーティションではいかなる頂点も訪問されないものと仮定して所定の予測方法により予測値を決定することを特徴とする請求項80に記載の多角形三次元メッシュ情報の復号化方法。

【請求項82】 三角形ツリー/三角形データ情報に含まれた三角形データ情報は、対応する三角形ツリーを構成する三角形別マーチ情報、幾何情報及び画像情報順に復号化されることを特徴とする請求項67に記載の多角形三次元メッシュ情報の復号化方法。

【請求項83】 三角形ツリー/三角形データ情報に含まれた三角形データ情報は、対応する三角形ツリーを構成する全ての三角形のマーチ情報、全ての三角形の幾何情報及び全ての三角形の画像情報順に復号化されることを特徴とする請求項67に記載の多角形三次元メッシュ情報の復号化方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 本発明はMPEG-4合成/天然混合符号化(Moving Pictures Expert Group-4 Synthetic and Natural Hybrid Coding: 以下MPEG-4 SNHCと称する)分野及び仮想現実モデリング言語(Virtual Reality Modeling Language: 以下VRMLと称する)で使われている三次元メッシュデータをエラー回復的(error resilience)、漸進的に復元(incremental build-up)する特性を有するように符号化/復号化する方法に関する。

【0002】

【従来の技術】 三次元メッシュからなる三次元客体の伝送においてはメッシュデータの効率的な符号化のみならず、伝送されるデータの漸進的な復元やエラー回復的な復元が重要な必要条件として認識されている。漸進的な復元においては伝送中の線路上のエラーによってデータ

50



13

の損失が発生した場合、既に伝送されたデータのみでも一部復元可能にすることによって、再伝送されるべきメッシュデータの量を最小化しうる。エラー回復的な復元では伝送された特定単位データ内のエラー発生有無とは関係なく伝送された単位データ別に独立して復元可能にし、データ復元効率を高めるだけでなく、使用者の待ち時間も短縮しうる。このように線路エラーに対して強い特徴のため、今後の無線通信または低伝送率通信等において効率よく使用可能な技術である。本発明は三次元メッシュデータの漸進的な符号化及びデータ損失に対する回復性を与えるために連結情報(Connectivity)、幾何情報(Geometry)及び画像情報(Photometry)からなる三次元メッシュデータを符号化/復号化する方法を提示する。

【0003】既存の三次元メッシュデータに対する符号化はメッシュデータ全体に対して順次に符号化することによって、符号化されたデータの伝送時全体のビットストリームを全て伝送される前に一部のデータのみで復元することは不可能であった。また、伝送時発生する線路上のエラーにより一部のデータのみが損失しても全体のデータを再び伝送すべき問題があった。例えば、現在MP EG-4 SNHC三次元メッシュ符号化技術として採択されているIBM社の符号化方式がある(ISO/IEC JTC1/SC29/WG11 MPEG98/W2301, "MPEG-4 SNHC Verification Model 9.0")。

【0004】MPEG-SNHCではVRMLに基づいてメッシュ符号化/復号化方法をデザインしている。VRMLにおいてメッシュはIndexedFaceSetというノード型で記述される。メッシュデータをコーディングするための主な技術としてIBM社の位相幾何学的サーザリ(Topological Surgery)がある。この技術は与えられたメッシュを球と位相幾何学的に同一であると仮定した後、与えられた切断エッジ(Cutting-edge)に沿ってメッシュをカットすることによって、2進ツリー構造の三角形スパニンググラフを生成する。この際、メッシュをカットするために定められた切断エッジは頂点と頂点を繋ぐ形態であって、ループを有するツリー構造として与えられる。これを頂点スパニンググラフと称する。従って、この二つのツリーを符号化/復号化すれば元のメッシュを損失なく再生しうる。

【0005】MPEG-4 SNHCでは、一つのVRMLファイルには多数のIndexedFaceSetが有り得るが、基本的に一つのIndexedFaceSetのみを基準として圧縮を行なう。しかし、一つのIndexedFaceSetは多数個のメッシュで構成される。即ち、任意の頂点と他の任意の頂点とを連結するエッジが元のメッシュに存在する多角形の集合を一つの連結成分(connected component)とすれば、IndexedFaceSetは多数個の連結成分で構成されることを許す。

【0006】一般に、グラフィックの迅速な処理のためには、モデリングされる単位が三角形であり、このような三角形がランダムに構成されるものでなくストリップやファンの形で相互連結されていることが望ましい。ま

(8)

特開2000-194843

14

た、シンボルを繰返して表現することにより圧縮率が良くなる。IBM社の符号化方式ではこのような圧縮率とグラフィックの迅速な処理という観点でメッシュを可能な限り一つの長い三角形ストリップで作る。

【0007】図1(A)乃至図1(F)はIBM社の三角形スパニンググラフを頂点スパニンググラフに沿ってカットして生成する位相幾何学的サーザリ技術過程を示す。図1(A)及び図1(D)はメッシュを太線で定義された切断エッジに沿ってカットする方法を示したものであり、図1(B)は切断エッジの全体形態を、図1(E)は図1(B)に沿ってメッシュをカットする時発生するエッジと頂点の構成形態を、図1(C)はカットする基準点の頂点を連結させて構成した頂点スパニンググラフ、図1(F)は頂点スパニンググラフに沿ってメッシュをカットして一つの連結された三角形の集合のストリップを定義した三角形スパニンググラフを示したものである。さらに、三角形スパニンググラフを図1(A)乃至図1(F)のような方法で生成するなら、一般に三角形スパニンググラフの分岐(branch)される二つのラン(run)のうち一方のランは他方のランに比べて相当短く表現されうる。

【0008】図2(A)乃至図2(D)は実際のメッシュに位相幾何学的サーザリ技術を適用した例を示した図面である。一方、頂点スパニンググラフは一本のブランチから多数本のブランチに分けられ、図3は頂点ランが再び以前の頂点のうち一つの位置に戻る形態、即ちループを有する頂点スパニンググラフの一例を示す。そして、一つのメッシュは多数個の連結成分で構成でき、実際にはメッシュをなす構成成分当り図1(C)のような頂点スパニンググラフと図1(F)のような三角形スパニンググラフの対を生成する。一般に、一つのIndexedFaceSetは多数個の連結成分で構成されるので、一つのIndexedFaceSetをコーディングすれば多数個の三角形スパニンググラフと頂点スパニンググラフの対が得られる。

【0009】位相幾何学的サーザリ技術により符号化されたデータの復元方法は次のような順序で構成される。

1. 頂点スパニンググラフを用いてバウンディングループを生成する。
2. 三角形スパニングツリーにおいて分岐する三角形の第3の頂点をY-頂点とするが、三角形スパニンググラフのビットストリームを用いてY-頂点を計算し、ストリップ形態の一連の連結性を有する三角形または多角形の集合を構成する。この際、三角形スパニンググラフの三角形マーチビットパターン(triangle marching bit pattern)を用いて三角形または多角形を生成する。

【0010】3. 前記1と2の過程を通じて構成されたツリー構造のメッシュ情報を用いて位相幾何学的構造によって復元されたメッシュで構成する。IBM社は頂点スパニンググラフと三角形スパニンググラフとをエントロピー算術符号化を用いて無損失圧縮している。しかし、IBM社の方法において元の構造を再構成するためには全ての

15

ビットストリームを入力すべきであり、これによって次のような短所を有することになる。

【0011】1. メッシュデータの復号化のためには全てのビットストリームを入力すべきなので、伝送中にエラーが発生した場合、全てのビットストリームを再伝送すべきである。

2. データの量が帯域幅(band width)に比べて多い場合伝送時間が長くなって使用者の待ち時間が延びる。使用者の待ち時間を短縮するためには既に伝送されたデータを活用してメッシュを部分的に復元、レンダリング(rendering)すべきであるが、従来の技術ではビットストリームの構造上伝送されたビット量に比べて復元可能な三角形の数が少ない。

【0012】このような従来の技術の短所を克服するためには次のような問題点を解決すべきである。

1. 伝送路の帯域幅や復号化器の特性を考慮してメッシュデータを効率的な単位に小さく分け、この単位のビットストリームが復号化器において復元及びレンダリング可能となるようにエラーに対する回復性を提供すべきである。

【0013】2. 現在受信している一部のデータのみでも一部の復元とレンダリングまで可能となるような漸進的な形態の符号化/復号化方法を提供すべきである。従来のIBM方法の基本構造上で前記両機能を提供する方法は、まず図4に示したようなバウンディンググループとY-頂点を処理する方法によってその性能が変わる。図1(F)において番号10、14、18に該当する点がY-頂点である。このY-頂点は分岐する両三角形ランのうち少なくとも一方のランが全て処理されてからでないと計算することができない。即ち、三角形ラン内に位置する三角形は、マーチビットパターンとバウンディンググループとを用いて三角形の三つの頂点に対するインデックスを決定しようが、分岐三角形の3番目の頂点のY-頂点は繋がる他の二つのランのうち少なくとも一方のランに対する全てのビットストリームが入力されてからでないと決定できない。従って、繋がるビットストリームが全て入力されるまでは分岐三角形の以降の三角形は復元、レンダリングできない。従来のIBMの方法では全てのビットストリームが復号化器に受信されていると仮定するので、かかる問題に対して考慮していないが、漸進的な方法で符号化/復号化するためにはかかる問題を解決すべきである。

【0014】以下、本発明の関連分野において用いられる用語を予め定義する。

▲多角形メッシュ(Polygonal mesh)：頂点の3次元空間上の座標(幾何情報)、それぞれの面とそれを構成する頂点との関係(連結情報)及びメッシュの幾何学的構造には影響を与えないが、どのように見えるかに対して影響を与える色相、法線、テクスチャ座標情報などの画像情報によって定義される。

(9)

特開2000-194843

16

【0015】▲フェース(Face)：一連の頂点インデックスの集合であり、コーナー(corner)は(面、頂点)の対を意味する。フェースを構成する頂点が相異なる頂点インデックスの集合である時、"simple face"と称する。

▲エッジ(Edge)：頂点インデックスの対をエッジと称する。もし、多角形メッシュにおいて1つのエッジが一面でのみ現れるとこれを"boundary edge"と定義する。もし、多面に同一なエッジが現れればこのエッジは"singular"であると定義する。但し、二枚の隣接面にのみ現れるエッジは"internal"であると定義する。boundary edgeとinternal edgeは"regular"であると定義する。

【0016】▲デュアルグラフ(dual Graph)：メッシュの各面の内部に各々1点を定義し、面間のinternal edgeを通過して前記定義した点を連結したグラフをデュアルグラフと称する。図5(A)は多角形メッシュを例示したものであり、図5(B)は図5(A)に示された多角形メッシュグラフのデュアルグラフを示したものである。

▲仮想現実モデリング言語(Virtual Reality Modeling Language: VRML)：インターネットで仮想空間を記述して伝送するために作られたグラフィック標準フォーマットである。

【0017】▲MPEG(Moving Picture Expert Group)：ビデオなどの多様なメディアの伝送のための圧縮フォーマットを標準化するための国際標準化活動である。

▲メッシュ(Mesh)：客体を多数個の多角形で構成して表現したものである。

▲ノード(Node)：頂点スパンニンググラフにおける一つの頂点、またはVRMLにおいて使用する描写の最小単位である。

【0018】▲位相幾何学的サーザリ(Topological Surgery)：メッシュを三角形のストリップ状に作るために与えられた経路(path)に沿ってメッシュをカットするIBM社のメッシュコーディングのための方法である。

▲頂点スパンニンググラフ(Vertex spanning graph)：位相幾何学的サーザリにおいてメッシュをカットする経路である。

【0019】▲三角形スパンニングツリー(Triangle spanning graph)：位相幾何学的サーザリにおいて頂点スパンニンググラフに沿ってカットして示す三角形ストリップ(triangle strip)であって、2進ツリー構造である。

▲vlast：頂点ランは上位頂点ランで多数本ブランチに分岐することがあるが、現在の頂点ランが該ブランチ中の最後のブランチなのかを示す。ブランチ中の最後であれば1、そうでなければ0が与えられる。

【0020】▲vrun：頂点ランが唯一の次のノードを有するかを示す。

▲vleaf：現在のvrunの端がリーフ(leaf)で終わるか、或いは分岐ノードで終わるかを示す。リーフで終わると1、そうでなければ0が与えられる。

50

(10)

特開2000-194843

17

▲vlength: 頂点ランの長さである。

▲loopstart: 頂点ランのうち一部はリーフが再び他の頂点ランに会う場合が発生するが、そうした場合の開始を示す。

【0021】▲loopen: 頂点ランのリーフがループを形成する場合、その終点を示す。

▲loopmap: loopstartとloopenの相互連関性を示す。loopenからloopstartへのエッジを関連付けるインデックスの集合である。

▲trun: 連続される三角形の集合であって、その端はリーフ三角形または分岐する三角形からなる。

【0022】▲tleaf: 三角形のランがリーフ三角形で終るか、分岐する三角形で終わるかを示す。リーフ三角形ならば1、そうでなく分岐する三角形ならば0が与えられる。

▲tmarching: 三角形の前進様相を記述する。ストリップにおいて右側境界にエッジを有すると1、左側境界にエッジを有すると0が与えられる。

【0023】▲polygonedge: 現在のエッジが元のメッシュモデルから与えられたエッジなのか、それとも多角形を三角形の集合として表現するために入れたエッジなのかを示す。元の与えられたエッジならば1、そうでなければ0が与えられる。

▲triangulated: メッシュ内に多角形の存否を表示する。存在すれば0、そうでなければ1が与えられる。

【0024】▲nvertices: 頂点の個数を示す。

▲nloops: ループの個数を示す。

▲nvedges: vrunの大きさを示す。

▲nvleaves: 頂点スパニンググラフにおいてリーフの個数を示す。

▲bitsspernvedges: nvedgesのために使われたビット数である。

【0025】▲simple: 頂点スパニンググラフがループを有すると0、そうでなければ1が与えられる。

▲ntriangles: 三角形スパニンググラフにおける三角形の数を示す。

▲ntbranches: 三角形スパニンググラフにおける分岐する三角形の数を示す。

▲nmarchingtrans, nmarchingkeepleft: tmarchingの圧縮のための統計的モデルである。

【0026】▲npolytrans, nkeeppoly: polygonedgeの圧縮のための統計的モデルである。従来のMPEGにおいて使用する三次元メッシュデータの圧縮方式は図6のようである。図6によれば、三次元メッシュデータ100は連結情報、幾何情報、そして画像情報に分れて各々連結情報符号化器102、幾何情報符号化器103、画像情報符号化器112を具備する符号化器101により符号化される。この際、頂点構造に対する情報105は連結情報符号化器102から幾何情報符号化器103に伝えられる。連結情報符号化器102、幾何情報符号化器103そして画像情報符号化器11

18

2で圧縮された情報はエントロピ符号化器104により圧縮されたビットストリーム111に置換えられる。

【0027】圧縮されたビットストリーム111は再び復号化器112に入力される。即ち、圧縮されたビットストリーム111はエントロピ復号化器106を経て連結情報及び幾何情報そして画像情報に分れ、これら情報は各々連結情報復号化器107、幾何情報復号化器108、画像情報復号化器113で各々復号化される。符号化器101と同様に頂点構造に対する情報109は連結情報復号化器107から幾何情報復号化器108に伝えられる。復号された連結情報、幾何情報、そして画像情報を用いて復元された三次元メッシュ110を構成しうる。

【0028】図6によれば、三次元メッシュは通信線路等で圧縮されたビットストリームの形態で伝送される。ところが、図6に示されたような従来のMPEGにおいて使用される三次元メッシュデータの圧縮方式はエントロピ符号化器104を使用するので、通信線路等で発生される伝送エラーに対して脆弱である。従って、伝送エラーに対応し、既に伝送された連結情報、位置情報、そして他の情報などを用いて漸進的に三次元メッシュを復元させる技術を本発明で提示する。

【0029】

【発明が解決しようとする課題】本発明は前記問題点を解決するために創案されたものであって、伝送中エラーが発生してもエラーの発生部分のみ再伝送したり、エラーの存する部分とは独立して残り復元するデータを復元することによってネットワークの負担と伝送時間を短縮でき、エラーの復元機能をネットワーク階層(Network Layer)に依存することで発生するビット量の増加を最小化し、伝送された一部の連結情報、幾何情報、そして特性情報を用いて三次元メッシュを漸進的に復元しうる三次元メッシュ情報の漸進的な符号化/復号化方法を提供することをその目的とする。

【0030】

【課題を解決するための手段】前記目的を達成するために本発明に係る多角形三次元メッシュを漸進的、エラー回復的に復元可能に符号化する方法の一実施例は、

(a) 多角形三次元メッシュを1つ以上の連結成分に分割する段階と、(b) 各連結成分に対して頂点グラフ情報及び三角形ツリー/三角形データ情報を生成する段階と、(c) 各連結成分に対し、その連結成分を構成する頂点グラフ情報及び三角形ツリー/三角形データ情報を、独立的に復号化可能な形式を有する基本メッシュの形式に合せて符号化する段階とを含むことを特徴とする。

【0031】前記目的を達成するために本発明に係る多角形三次元メッシュを漸進的、エラー回復的に復元可能に符号化する方法の他実施例は、(a) 多角形三次元メッシュを1つ以上の連結成分に分割する段階と、(b) 各連結成分に対して頂点グラフ情報及び三角形ツリー/三角形データ情報を生成する段階と、(c) 各連結成分に対し

10

20

30

40

50

(11)

特開 2000-194843

19

て、その連結成分を構成する頂点グラフ情報及び三角形ツリー/三角形データ情報を、独立して復号化可能に固定されたビットストリーム形式を有する基本メッシュの形式に合せて符号化する段階とを含むことを特徴とする。

【0032】前記目的を達成するために本発明に係る多角形三次元メッシュを漸進的、エラー回復的に復元可能に符号化する方法の他実施例は、(a) 多角形三次元メッシュを1つ以上の連結成分に分割する段階と、(b) 各連結成分に対して頂点グラフ情報及び三角形ツリー/三角形データ情報を生成する段階と、(c) 各連結成分に対して、その連結成分を構成する頂点グラフ情報及び三角形ツリー/三角形データ情報を、独立して復号化可能に情報の特性に応じて可変的なビットストリームの形式を有する基本メッシュの形式に合せて符号化する段階とを含むことを特徴とする。

【0033】前記目的を達成するために本発明に係る多角形三次元メッシュを漸進的、エラー回復的に復元可能に符号化する方法の他実施例は、(a) 多角形三次元メッシュを1つ以上の連結成分に分割する段階と、(b) 各連結成分に対して頂点グラフ情報及び三角形ツリー/三角形データ情報を生成する段階と、(c) 各連結成分に対してその連結成分を構成する前記頂点グラフ情報を符号化する段階と、(d) 前記三角形ツリー/三角形データ情報を分割したデータパーティションに各々仮想ビット対を追加することによって仮想連結成分で構成して符号化する段階とを含むことを特徴とする。

【0034】前記目的を達成するために本発明により多角形三次元メッシュを漸進的、エラー回復的に復元可能に符号化する時、三角形三次元メッシュをデータパーティションに分割してバケット化する方法の一実施例は、(a) 三角形ツリーに含まれた三角形を順次に訪問しながらその三角形における総ビット発生量を計算する段階と、(b) 前記(a)段階で計算された三角形の総ビット発生量を累積する段階と、(c) 前記(b)段階で累積した値がバケット大きさにバケット許容値をかけた値より小さな場合には三角形ツリーに含まれた次の訪問三角形に対して前記(a)段階以降を繰返し、小さくない場合には前記訪問された三角形までの三角形ツリー/三角形データ情報をデータパーティションに分割してバケット化する段階とを含むことを特徴とする。

【0035】前記目的を達成するために多角形三次元メッシュをエラー回復的に復号化する方法の一実施例は、(a) 入力されたビットストリームを基本メッシュ単位に区分する段階と、(b) 前記基本メッシュのパーティション類型を判断する段階と、(c) 前記基本メッシュに頂点グラフ情報が含まれていると、前記頂点グラフ情報を復号化してバウンディンググループテーブルを生成する段階と、(d) 前記基本メッシュに三角形ツリー/三角形データ情報が含まれていると、前記三角形ツリー/三角形デ

20

ータ情報を復号化して三次元メッシュを生成する段階と、(e) 前記(a)段階乃至前記(d)段階を繰返すことによって三次元メッシュを生成する段階とを含むことを特徴とする。

【0036】前記目的を達成するために多角形三次元メッシュをエラー回復的に復号化する方法の他実施例は、(a) 入力されたビットストリームを基本メッシュ単位に区分する段階と、(b) 前記基本メッシュのパーティション類型を判断する段階と、(c) 前記基本メッシュに頂点グラフ情報が含まれていると、前記頂点グラフ情報を復号化してバウンディンググループテーブルを生成する段階と、(d) 前記基本メッシュに三角形ツリー/三角形データ情報が含まれていると、連結成分単位で前記三角形ツリー/三角形データ情報を復号化して三角形三次元メッシュを生成する段階と、(e) 前記(d)段階における連結成分が仮想連結成分の場合前記(a)段階の以降を繰返し、そうでない場合三角形三次元メッシュの生成を完了する段階とを含むことを特徴とする。

【0037】

【発明の実施の形態】以下、添付された図面に基づき本発明を詳しく説明する。まず、三次元メッシュの漸進的な処理のために本発明では新たなメッシュ構造を図7のように提案する。図7によれば、三次元メッシュ(Mesh Object: MO)はメッシュの情報を多段階に分類した段階別メッシュ(Mech Object Layer: MOL)で構成される。この際、各MOLは1つまたはそれ以上の基本メッシュ(Mesh ObjectBase Layer: MOBL)を含むことになる。一つのMOBLはそれ自体の復元に必要な連結情報、幾何情報、そして画像情報などを含む。即ち、MOは符号化する三次元メッシュ客体単位で定義され、これらメッシュ情報の多様な画質と機能面で多段階に分類し、この分類された各段階を段階別メッシュ(MOL)として定義する。さらに、位相幾何学的サーザリ方法を用いて一つの三次元メッシュ客体を相互連結性のない多数個の独立したメッシュ情報(即ち、連結成分)で構成した時、符号化するデータの大きさやその他の特性に応じて相互独立したメッシュ情報を統合したり分割させて構成したものを基本メッシュとして定義する。

【0038】図8(A)及び図8(B)はMOとMOBLの例を示す図面である。このような新たなメッシュ構造でさらに効率よくメッシュ情報を漸進的に復元/レンダリングするためにはY-頂点を考慮したバウンディンググループのインデックス決定方法が必要である。一つの三角形メッシュを復元するためには、図9のように三角形ツリー/三角形データ(tt/td)対のパーティションに右側境界(right boundary)と左側境界(left boundary)の開始点に対するバウンディンググループのインデックスが与えられる。図9において、mkは右側開始インデックス(right starting index)、nlは左側開始インデックス(left starting index)、矢線はコーディングされる三角形の順序であり、こ

(12)

特開2000-194843

21

の時前進方向を基準として三角形ストリップの右側に位置する境界を右側境界、左側に位置する境界を左側境界、陰影がつけられた三角形は分岐三角形を示す。tt/t<sub>d</sub>対の各三角形に対してバウンディンググループのインデックスを決定するためには次のような付加情報が必要である。

#### 【0039】●分岐三角形以前の三角形の場合

図9に示されたように、分岐三角形以前の三角形の場合にはバウンディンググループのインデックスが左側境界では1ずつ増加し、右側境界では1ずつ減少するので、マー

#### ●分岐三角形の場合

図9において分岐三角形は三つの頂点b[m-3]、b[m-10]、b[n+2]で構成される。しかし、分岐三角形において右側ブランチの三角形情報、即ち三角形の個数が分かる情報が入力されないと、分岐三角形の3番目の頂点のY-頂点のバウンディンググループにおけるインデックスのm-10がわからない。従って、Y-頂点のインデックスを決定するためには一方のブランチの三角形の個数が分かる情報が入力されるべきである。一方のブランチの三角形の総数を知るためには、tt/t<sub>d</sub>対で三角形ランに対する情報が必要であるが、図9において、右側ブランチの三角形の数をp個と仮定すれば、用いられる頂点の個数を次の式①によって決定しうる。

#### 【0040】

1つのブランチにおいて使われる頂点の数 =  $p + 2 \dots$  ①  
例えば、図9において分岐三角形の3番目の頂点のY-頂点のバウンディンググループにおけるインデックスは右側境界上の頂点のインデックスがm-3であり、ブランチ内の三角形の数が6個という事実を用いれば、Y-頂点のインデックス =  $(m-3) - (6+2-1) = m-3-7 = m-10$ と決定しうる。

#### 【0041】●分岐三角形から分岐するブランチ内の三角形の場合

Y-頂点のインデックスが提供される場合には前記分岐三角形以前の三角形を処理する方法と同一な方法で各三角形を復元及びレンダリングできるが、分岐三角形のY-頂点が決定されないと三角形の頂点のインデックスが決定できない。これは、図9のように、右側ブランチはY-頂点が解決されないと左側境界上に位置する頂点のインデックスを前記理由によって決定できないからである。

【0042】このように分岐三角形が発生するY-頂点の問題を解決してさらに効率よく漸進的な復元及びレンダリングを提供するための方法を後述のように提示する。また、本発明ではエラーに対する回復性を提供するために、伝送路の帯域幅や復号化器の特性などを考慮してメッシュデータを分割するに当って、固定構造分割及び可変構造分割の2種の方法で各々処理でき、分割されたデータを各々相互独立して復元及びレンダリングするため

22

の方法をさらに提示する。ここで、固定構造分割は分割されたデータが全て同一な構造を有するように構成させる方法であり、可変構造分割は分割されたデータが符号化効率及びバケット化を考慮して頂点グラフ、三角形ツリー、三角形データなどの情報により適切に組合わせられた構造を有するように構成した方法である。

【0043】<Y-頂点を考慮した漸進的符号化方法>従来の位相幾何学的サーザリ方法では符号化の順序が全体tt/t<sub>d</sub>対に対して特定の方向にのみ固定されている。かかる方法で漸進的なレンダリング能のみを向上させるように符号化するためには、メッシュデータから発生する全てのY-頂点のインデックス値とY-頂点の総数の情報をビットストリームに含んで復号化器に伝送すればよい。しかし、この方法は符号化の効率面で非常に望ましくない方法である。従って、両側面、即ち漸進的なレンダリング効果及び符号化効率を満足するさらに効率的なY-頂点に対する符号化方法が要求される。

【0044】位相幾何学的サーザリ方法では従属ツリー間の長さ差が大きく、大きなメッシュデータであるほどY-頂点を効率よく決定することができず、多角形の復元による漸進的なレンダリング効果が期待できないという問題がある。従って、従属ツリー間の長さ差がある場合、即ち二本の分岐されたブランチの大きさが同一でなければ、Y-頂点の決定とレンダリング側面で短い長さの従属ツリーを先に符号化することがさらに効率的である。ここで、図2(A)乃至図2(D)に示された位相幾何学的サーザリ技法を用いてメッシュデータを得る場合、一般に一方のブランチの三角形の数が対応する他方のブランチの三角形の数より少なくなる。このような特性により、本発明では従属ツリーの大きさに応じる方向性(Orientation)情報を提供して符号化順序(traversing)を分岐三角形毎に個別的に定められる方法を提示する。即ち、1ビットで構成された方向性情報が1ならば最初に決まった符号化順序のように右側を先に訪問し、0ならば左側を先に訪問して符号化する。従って、このように方向性情報を復号化器に伝達することによって漸進的な復号化及びレンダリング効果を向上させることを期待しうる。さらに、方向性情報は任意の1つのメッシュツリーをメインツリーと従属ツリーとに分類することができ、この特性を用いて連結性情報を保たせながらさらに効率よくデータを分割しうる機能も提供する。

【0045】図10(A)乃至図10(C)は前記方法により決定した方向性情報を活用した符号化方法の一例を示す図面である。図10(A)において陰影三角形はメインツリーに存在しながら方向性情報を有している分岐三角形を、2進メッシュツリー内部の線は従属ツリーを有している2進メッシュツリーのメインブランチを、そして2進メッシュツリーの外部線は従属ツリーにおける符号化順序、即ち復号部においてバウンディンググループのインデックスをマッピングする方向を示す。ここで、バウンディン

					特開2000-194843
		23	(13)		24
		グループのインデックスをマッピングする方向は、図10(A)の例のように方向性情報と同じ方向(例えば、0なら時計回り方向、1なら逆時計回り方向)に決定され、各従属ツリー毎に変えて定義しうる。これとは異なって、バウンディンググループのインデックスをマッピングする方向は、方向性情報とは関係なく全て時計回り方向、または全て逆時計回り方向のみでも定義しうる。一方、復元する従属ツリー内の三角形の数をt個と仮定すれば、Y-頂点値は方向性情報によって次のように計算式が異なる。			数の使用者による共有を可能にする。一方、バケットが符号化の結果として得たビット列を一定の長さ毎に束ねたものであれば、パーティションは符号化しようとする元の情報(メッシュ)を適当な単位に分割したものを意味する。この分割過程では、受信側で復元する時伝送路上のエラーに効率よく対応できるように考慮する。即ち、全体のメッシュを意味のある部分に分割すれば、符号化されたビット列が伝送路上でエラーを含んで受信側で完璧な復元が出来なくてもエラーが含まれたパーティションのみが損傷されるだけで、エラーの含まれない他のパーティションは完璧に復元しうる。ここで、意味のある部分とは、例えば動物の絵において、腕、足、胴体等、各部分が相互関連性のある意味を有することをいう。パーティションをする過程では、バケットのようにその大きさが一定に決まっているわけではない。但し、受信されたデータにエラーが含まれた場合にも、受信側で効率よくデータを復元することが可能なパーティション方法を本発明では提示する。
		【0046】●右側ブランチを先に符号化する場合(方向性情報値が1の場合)	10		【0048】前述したように符号化するデータをパーティションに分割することによって次のような効果を期待しうる。
		分岐三角形の右側境界上の頂点のバウンディンググループにおけるインデックスがpならばY-頂点のインデックスは次のようである。Y-頂点の位置=p-t-1			●損失が発生したデータ単位のみ再伝送させることによりネットワーク負荷及び復号化遅延時間を縮めることができる。
		●左側ブランチを先に符号化する場合(方向性情報値が0の場合)			●分割されたデータ単位間に相互独立性を保障するならば、
		分岐三角形の左側境界上の頂点のバウンディンググループにおけるインデックスがpならばY-頂点のインデックスは次の通りである。Y-頂点の位置=p+t+1	20		1) 損失されたデータの再伝送無しでも、損失のない残りデータのみでメッシュの復元及びレンダリングが可能であり、損失によって発生する復号化遅延時間を除去しうる。
		図10(B)と図10(C)においてtrは三角形ラン、tlはtleaf、tdは方向性情報、tmはマーチ(marching)を示し、図10(B)はtt/td対情報の構成要素のうち三角形ツリーに方向性情報を配置する場合であり、図10(C)はtt/td対情報の構成要素のうち三角形データに方向性情報を配置する場合のビットストリームの構成図を示す。図10(B)に示された配置方法は具現が簡単な長所があり、図10(C)に示された配置方法は1つの三角形の情報が復元されると直ちにレンダリングできて遅延時間が短く、復号化器の設計時メモリの使用が図10(B)に比べて少なくコストダウンとなる長所がある。	30		【0049】2) メッシュデータ全体を受信していない状態でも、現在受信したデータを順次に復元及びレンダリングしうる。
		【0047】図11(A)は従来の位相幾何学的サーザリの符号化順序を示し、図11(B)は分岐する三角形における両側ブランチの大きさに応じる符号化順序の決定方法を適用した一例であって、方向性情報の意味を示した図面である。データ分割時の方向性情報決定方法は後述する。			3) データ単位間に相互独立性が保障されない符号化方法に比べて、損失されたデータが再伝送されない場合のメッシュ復元能がさらに優れる。また、再伝送がある場合にも伝送効率(復号化待ち時間)が向上する。
		<エラーに対する回復性を考慮した漸進的な符号化方法>			【0050】<固定構造分割方法>図12(A)乃至図12(I)は固定構造を有するデータ分割方式を示す図面である。sdはバケット化されたメッシュビットストリームにおいてパーティションの開始位置を示す開始コードであり、idはパーティションの識別子であって、本発明に係る符号化文法の一実施例では、各々3D_MOBL_start_code及びmobl_idで表される。vq_idは多数個の連結成分で構成されたメッシュデータにおいて多数個の頂点グラフ情報を束ねて処理する場合、各tt/td対のパーティションの復元時これに対応する頂点グラフを指定するために使用する表示子であって、本発明に係る符号化文法の一実施例ではcodap_vq_idで表される。訪問インデックス(visiting index: vi)は分割されたパーティシ
		データ伝送路や符号化器から生成されたビットストリーム上のエラーによるデータの損失発生時、復号化器の方ではこれに対する効率的な処理方法が必要である。従って、符号化するメッシュデータを意味のある単位(即ち、パーティションまたは基本メッシュ)に区分してデータを分割し、これを所定の大きさを有する処理単位(以下、バケットと略称する)で伝送することになる。具体的に、バケットは特定の形式に配列されるビットの束であって、一定の長さを有するように決まっている。伝送路上に長いデータを伝送しようとすれば、多数の使用者が存在する伝送路上で停滞を引起すので、伝送単位を一定の長さ	40		ックス(visiting index: vi)は分割されたパーティシ
		に束ねることにより限定された伝送路を多	50		

(14)

特開 2000-194843

25

ンのルート三角形のバウンディングインデックス (bound  
ing\_index) 値を示す表示子であって、図4のように left  
\_index と right\_index 情報を有しており、本発明に係  
る符号化文法の一実施例ではこれらが各々 codap\_left  
\_idx, codap\_right\_idx で表される。bp は境界予測 (b  
oundary\_prediction) を示す表示子であって位置 (geome  
try)、色相 (color)、法線 (normal)、テクスチャー座標  
(texcoord) 情報を符号化する方法を決定する。即ち、bp  
が 1 ならば既に符号化した頂点に対して重畳を許して符  
号化する方法を意味し、0 ならば重畳されないように頂  
点を符号化する方法を意味する。bp は本発明に係る符号  
化文法の一実施例において codap\_bdry\_pred で表され  
る。1bl は現在のパーティションが分岐三角形で終わる  
場合に前記分岐三角形の Y-頂点インデックスを現在のパ  
ーティション情報のみでも復号化器で計算可能にするた  
めに発生する表示子であって、前記分岐三角形において  
任意の一方のブランチの従属ツリーの総大きさを示す。  
1bl は本発明に係る符号化文法の一実施例において codap  
\_branch\_len で表される。vql は頂点グラフ、tt は三角  
形ツリー/三角形データにおける三角形ツリー、そして t  
d は三角形ツリー/三角形データにおける三角形データを  
示す。また、1vql は 1 つのパーティション内で復元する  
連結成分の存否を示すために定義された 1 ビット表示子  
であって、次に復元する vql が存在する場合には 0、存在  
しない場合には 1 と定義される。1tql は 1 つのパーティ  
ション内で復元する連結成分または従属ツリーの存否を示  
す 1 ビットの表示子であって、その値が 0 なら存在する場合を、1 なら存在しない場合を各々示す。固定構造分割  
方法は図 12(A) 乃至図 12(I) に定義されたビットストリ  
ームの形態のうち適用分野に応じて任意の 1 つの形態の  
みで全体メッシュデータを同一に符号化する方法であ  
る。

【0051】構造面で図 12(A) は全体メッシュデータを  
符号化順序に応じて処理し、一つのパーティションで構  
成する方法を示したものである。図 12(B) は図 12(A) の  
ように全体メッシュデータを一つのパーティションで構  
成するが、頂点グラフデータと tt/td 対を分離して構成  
する方法を示した図面である。図 12(C) は一つの連結成  
分を最小単位でパーティションを構成する方法を示した  
図面である。図 12(D) はメッシュデータを構成する多数  
個の頂点グラフを集めて一つのパーティションに構成  
し、また対応する tt/td 対を集めて一つのパーティシ  
ョンに構成する方法を示した図面である。図 12(E) は多数  
個の頂点グラフを集めて一つのパーティションに構成  
し、tt/td 対はパーティションの大きさを考慮して多数  
個のパーティションに分割して構成する方法を示した図  
面である。図 12(F) は図 12(D) のように頂点グラフと tt  
/td 対を分離して構成し、これらを連結成分単位別に各  
々別のパーティションに構成する方法を示した図面であ  
る。図 12(G) はメッシュ客体を構成する多数個の頂点グ

26

ラフを各々別のパーティションに構成し、次いで対応す  
る全ての tt/td 対をパーティションの大きさを考慮して  
多数個のパーティションに分割して構成する方法を示し  
た図面である。図 12(H) は各連結成分を構成する頂点グ  
ラフを先にパーティション化し、対応する tt/td 対をパ  
ーティション化して全ての成分を符号化する方法を示し  
た図面である。図 12(I) は各連結成分において頂点グラ  
フデータを先に一つのパーティションで符号化し、対応  
する tt/td 対をパーティションの大きさを考慮して分割  
してパーティション化する方法を示す。

【0052】機能面で図 12(A) 乃至図 12(I) のそれぞれ  
の方法を説明する。まず図 12(A) と図 12(B) に示された  
方法はエラー回復性が提供されなかったり、伝送路や復  
号化器の性能に対する制約がない場合に用いられる方法  
であって、図 12(A) は図 12(B) に比べて処理費用とレン  
ダリング効率はさらに向上されるが、データの損失時に  
発生する再伝送費用はさらに多くかかる。図 12(C)、図  
12(D)、図 12(F) 及び図 12(H) に示された方法では、パ  
ーティション毎に相互独立性は保障されるが、特定大き  
さの単位のバケットに合せにくい。即ち、特定大きさの  
バケットに載せて伝送する場合、パーティションがバケ  
ットより小さい場合、その差だけのダミー (Dummy) 情報  
を追加する必要がある。逆の場合には、バケット化する  
ことが出来ないため、これに対する別の処理方法及び付  
加情報が復号化器に提供されるべきである。従って、全  
体符号化効率が低くなり、複雑度及びコストが増加する  
問題がある。そして、図 12(E)、図 12(G) 及び図 12(I)  
に示された方法の場合、メッシュの連結性情報を提供す  
る頂点グラフデータは前記方法と同様に処理されるが、  
メッシュにおいて三角形 (多角形) の構成形態を提供す  
る tt/td 対は相互独立性を保障させながらもパーティシ  
ョンの大きさに応じて分けて処理できる。tt/td 対のパ  
ーティションには vq\_id、vi と bp 情報が追加される。こ  
こで、vq\_id は復元する tt/td 対に対応する頂点グラフ  
を指定する指定子であって、図 12(D) 乃至図 12(G) のよ  
うに頂点グラフを束ねて処理する場合に使われる。vi は  
tt/td 対が多数個のパーティションに分けられた時、各  
パーティションの開始三角形における left\_index と rig  
ht\_index 値を指定する指定子であって、以前のパーティ  
ションが損失によって復元不可能な場合でも現在のパ  
ーティションを以前のパーティションとは独立して復元  
可能にするために提供される情報である。さらに、bp は  
頂点に対する位置、色相、法線、テクスチャー座標情報  
を符号化する方法を定義する 1 ビット表示子であって、t  
t/td 対の各パーティション毎に定義される。また、図 12  
(E)、図 12(G) 及び図 12(I) に示された方法において 1b  
1 は分岐三角形においてパーティションが終わった場合  
にのみ選択的に使用されるように構成される。図 12(A)  
乃至図 12(D)、図 12(F) 及び図 12(H) に示すように、連  
結成分単位で tt/td 対を符号化する場合には bp を 1 とし

(15)

特開 2000-194843

27

て重畳が許容される方法で復元される。一方、図12(E)、図12(G)及び図12(I)に示されたように任意の1つの連結成分を構成するtt/td対が多数個に分割される場合には各パーティション毎にbp表示子を定義して復元させる。

\*

right\_index-left\_index-1=現在のパーティションにおける三角形の数…②

前記式②を満たせばブランチの端を意味するのでltqを1で符号化することになる。

【0054】このような構造を提供するためのデータ分割の方法は次の通りである。

#### 1. 連結成分単位でデータを分割する方法

図12(C)及び図12(H)の場合であって、この方法は具現が簡単で連結成分間の大きさ差がなければ効率的である。しかし、連結成分のデータ大きさが均一でなく差が大きいと、データパーティションの大きさが一定しなくてこれに対する付加的なビットが発生して符号化効率を低下させる。一方、連結成分自体のデータ大きさが大きければ符号化効率を低下させるだけでなく、所定の大きさにバケット化できない。従って、この方法は伝送路や復号化器の性能に制約がない時のみ制限的に適用可能な方法である。一方、図12(A)、図12(B)、図12(D)及び図12(F)の場合も前記問題を有している。

#### 【0055】2. 小さい連結成分を合わせてデータを分割する方法

小さい連結成分を多く含むメッシュでは前記1の方法を適用することは符号化効率面で適切な方法ではない。従って、図12(A)及び図12(B)の場合のように小さい連結成分は所定の大きさのパーティションを満たすまで集めて一つのパーティション単位内で処理することが望ましい。この方法において、パーティションのヘッダ情報に特定の頂点グラフに対応するtt/td対を定義するための指定子(vq\_id)を提供すべきである。

【0056】3. 大きな連結成分をデータ分割する方法  
前記1の方法を使用する場合、バケットより大きい連結成分を符号化することは制限的な環境でのみ可能であった。従って、本発明ではパーティション大きさに対する柔軟性とパーティションの相互独立性を全て満たすように、大きな1つの連結成分を分割しうるさらに一般的な符号化方法を提示する。図12(E)、図12(G)及び図12(I)は任意の大きな連結成分からtt/td対のみを再び多数個のパーティションに分割して符号化処理する方式を示す。図12(E)は図12(D)から、図12(G)は図12(F)から、図12(I)は図12(H)から各々変形された形態を示す。一方、図12(E)及び図12(G)は、図12(D)及び図12(F)でのように全体の頂点グラフに対する情報を復号化器で全て貯蔵すべきなのでコスト高となる短所を、図12(I)は、符号化効率面で小さな連結成分に対して負担が大きいとの短所を有している。

【0057】さらに、この方法では特定の頂点グラフに対応するtt/td対を定義するための指定子(vq\_id)とバ

28

\*【0053】ltqは選択的方法で符号化される。即ち、現在のパーティションの情報がtt/td対の何れか1本のブランチの最後であれば1で符号化し、そうでなければ0で符号化する。これは訪問インデックス(visiting index)を用いて判断し、次のような公式を用いる。

パーティションの開始三角形の境界ループインデックスを指定するための指定者(vi)及び幾何、色相、法線、テクスチャー座標情報等の符号化する方法を表示する表示子(bp)情報がパーティションヘッダ情報にさらに定義されるべき負担を有している。しかし、この方法はメッシュデータを損失から相互独立して容易に保護でき、パーティション化に伴う符号化損失も減らしうるさらに一般的な分割方法である。

【0058】さらに、図13(A)は、tt/td対の分割時のビットストリームの構造を示す例である。この中で、垂直点線は分割された位置を表示したものであって三角形ツリーと三角形データとの相互関係を示す。図13(B)は前記方法を適用してデータを分割する過程を示した例を示す。図13(B)においてid情報は便宜上省略したが、開始コード(start code)に続いて定義される。以下、vqは頂点グラフ、tt/tdは三角形ツリー/三角形データ、ccは連結成分を各々略称する。図13(B)においてnは連結成分の個数、n'は連結成分を大きさに応じて再構成した場合の構成成分の個数である。再構成の方法は、

#### 1. 連結成分の再合成(251段階)

小さな連結成分は合わせてデータパーティションの大きさの範囲に入るようにする。図13(B)ではcc<sub>1</sub>とcc<sub>2</sub>を合わせてcc'<sub>1</sub>を作る。一方、データパーティションを考慮してこの大きさに比べて大きなものは内部で分割して小さな仮想構成成分に分割する。図13(B)の例ではcc<sub>3</sub>を多数子のcc'<sub>1</sub>、...、cc'<sub>k</sub>に分割した。

#### 【0059】2. vqとtt/td対の生成(252段階)

再構成した各構成成分に対してvqとtt/td対を作る。

#### 3. vqとtt/td対の統合(253段階)

前記252段階で生成したvqとtt/td対を各々別に統合する。

#### 4. vq情報の分割(254段階)

前記253段階で統合したvq情報を適切な大きさの単位にデータ分割する。

#### 【0060】5. tt/td対情報の分割(255段階)

前記253段階で生成したtt/td対情報を適切な大きさの意味のある単位に分割する。このような固定構造分割方法は図12(A)乃至図12(I)の各方式毎にその構造を支援しうる復号化器が全て必要なので、実際にシステムを構成する側面では複雑度とコスト面で不都合な面を有している。従って、一つの復号化器のみでも前記方式のビットストリームを全て適応的に復号化可能にするために本発明では可変構造分割方法を提示する。

【0061】<可変構造分割方法>可変構造分割方法で



(16)

特開 2000-194843

29

30

は固定構造分割方法で提供しないpt(partition type)情報、即ち現在のパーティションの分割された形態を指定する情報が追加され、図14(A)乃至図14(D)に示すように、総4種の構造に分類される。本発明で定義したptはs\*

\* c値によって定義され、scとpt値との関係は表1のようである。

【0062】

【表1】

ビット値	内容
0000 0000 0011 0001	1つまたは多数個のvgを含み、このvgデータは他の1つまたは多数個のデータパーティションで使われる。
0000 0000 0011 0011	tt/td情報のみを含み、この情報を復元する場合任意のvgにtt/td対を対応させるvg_idと三角形ストリームの開始点のviとbpをヘッダに含む。
0000 0000 0011 0100	vg、tt/tdが一つのデータパーティションを形成し、vgは他の1つまたは多数個のデータパーティションで使われる。

可変構造分割方法はコストと複雑度面を考慮するための方法であって、次の事項を考慮する。

▲開始コード(start code)は選択的に適用されない。即ち、復号化器は現在のビットストリームの形態がどんなモードで符号化されたのかを知らない状態で復号化する。従って、開始コードを符号化器で符号化モードに応じて選択的に提供する場合、制限されたビットストリームのみを復元することになり、場合に応じては多種の復号化器を提供すべきである。これにより、コストと複雑度が増加する結果を招く。つまり、符号化器の符号化条件にも拘らず復号化するためには可変的な文法体系は排除すべきである。

【0063】▲特定の1つの形態のみで符号化する場合、メッシュの特性や復号化器の特性により不要な情報が提供されるので符号化効率が低くなることがある。図14(A)乃至図14(D)は可変構造を支援しうる4つの文法モジュールを示した図面である。図14(C)は図12(A)乃至図12(I)において定義されたように、vq\_idとltqを適応的に提供する文法を有する。即ち、図14(B)の文法においてvqが1つだけで構成される場合、復元過程でvqとtt/td対の対応関係が分かるのでvq\_idの提供は不要である。ltqの場合、分割されたtt/td対が全体tt/td対のブランチの端部に該当される時のみこれを表示するために提供される。これに対する判別式は式②のようである。図14(A)はpt=0であり、任意の個数の連結成分でパーティションを構成する構造を示す。図14(B)はpt=1であり、任意の個数のvqだけでパーティションを構成する構造を示す。

【0064】図14(C)はpt=2であり、一つの連結成分に対応するtt/td対のみをパーティションで構成する構造を示す。図14(D)はpt=3であり、一つの連結成分に対応するvqとtt/td対をパーティションで構成する構造を示す。従って、このような文法構造の組合は図12(B)を除く

※いた残り全ての形態の文法構造を支援しうる。即ち、図12(A)の場合にはpt=0で具現可能で、図12(I)の場合にはpt=1とpt=2の組合で具現可能である。従って、本発明では符号化効率側面を考慮してパケットの大きさに比べて相対的に小さな連結成分はpt=0を使用することによって1つのパーティション内に多数個の連結成分を集めて符号化し、パケットの大きさに比べて相対的に大きな連結成分の場合にはpt=1とpt=2またはpt=2とpt=3を使用して符号化する。また、伝送路や復号化器の能力側面を考慮する場合、これに適切に様々に組合わせられた多様な形態の文法も提供しうる。

【0065】図15(A)乃至図15(D)は可変構造分割方法を適用した例である。図15(A)において、cc<sub>1</sub>乃至cc<sub>3</sub>は小さなメッシュなので、pt=0またはpt=3と定義して連結成分別に処理する。cc<sub>4</sub>は大きなメッシュなので、tt/td対を多数個に分割してpt=3とpt=2を組合せた形態で処理する。さらに、cc<sub>4</sub>の場合、図15(B)のようにpt=1とpt=2を組合せた形態でも処理しうる。また、図15(C)に示されたように、符号化効率を向上させるためにはメッシュデータに対して他の組合わせ構造に変形することもできる。図15(C)はcc<sub>1</sub>乃至cc<sub>3</sub>のメッシュデータはpt=0と定義した場合を示したものであって、全体メッシュデータを表現するために図15(C)と図15(A)が結合された形態または図15(C)と図15(B)が結合された形態でも表現しうる。但し、図15(C)の構造を提供するためには最後の連結成分を判別しうる条件が必要である。従来では最後の連結成分を決定するためにlast\_ccという1ビット表示子が使われるが、1なら最後の連結成分を意味する。従って、本発明ではlast\_ccと類似した機能をするltq表示子とsc内に存するpt情報を用いて復元する連結成分の存否を次の通り判別しうる。

【0066】

```

if(ltq==1)[
  if(pt==0)[
    if(next_2_bytes()==sc) 現在復元されたccはパーティション内の最後のccである;
  else
    現在復元されたccは復元する最後のパーティションである;
  ]
]

```

(17)

特開 2000-194843

31

]else

cである;

]else

cではない;

next\_2\_bytes()は復号化器においてビットストリームのうち2バイトのみ予め読出す関数である。図15(D)はp<sub>t</sub>=0, 1, 2を用いた実施例を示している。前記例から分かるように、可変構造分割方法は符号化環境に対する適応性に優れ、符号化効率やコスト面で固定構造分割方法より向上された方法である。

【0067】<tt/td対のデータ分割方法>本発明では、tt/td対の分割のために以前パーティションの端部から最初に現れる分岐三角形の何れか一方のブランチの三角形情報が全て入力された後の位置(=メインブランチ)で、データ分割することを基本とする。図16はこのように例を示す。このようにメインブランチでtt/td対を分割することは、前述したように、Y-頂点のインデックスが決定できないと該分岐三角形以降の三角形の復元及びレンダリングができないからである。しかし、この方法は最初に決まった訪問順序と実際のtt/td対の形態に応じて非効率的な場合が発生する。即ち、左側(或いは右側)ブランチコーディングが優先する場合、tt/td対の分岐三角形の左側(或いは右側)ブランチが非常に大きければ相変らずバケットの大きさを越え、Y-頂点のインデックスを決定することは分岐三角形以降の多量の情報、即ち一方ブランチの三角形数に対する情報が処理されるまで待たなければならない。

【0068】従って、本発明では効率よくメッシュデータを分割する方法に対して3つの方法を提示する。本発明ではバケット内に1つ以上のパーティションが含まれることもある。しかし、ここでは一つのバケット内に一つのパーティションのみが含まれるという仮定下で説明する。第1方法は、図17に示すように、ステップ300において取り込んだバケットの大きさ(=target)に対してバケット許容値(=t<sub>rate</sub>)を設定して分割する方法であって、連結成分の単位で符号化するttの平均ビット発生量(=tt<sub>mean</sub>)を求め、予め設定したバケット許容値(=t<sub>rate</sub>)を満たす範囲内で実際にデータ分割を行う。図17において、ステップ310のtt<sub>mean</sub>は連結成分別に計算したttの平均ビット量であり、ステップ320のtt<sub>mean</sub>、td<sub>bits<sub>i</sub></sub>はi番目の三角形が符号化された時vqとtt情報を除いた残り情報のビット発生量であり、ステップ320のcur<sub>bits</sub>はバケットの最初の三角形からi番目の三角形まで発生したビット発生量(tt<sub>mean</sub>+td<sub>bits<sub>i</sub></sub>)を全て合せた値である。従って、i番目までの総ビット発生量(cur<sub>bits</sub>)が許容されたバケット大きさ(=t<sub>rate</sub>\*target)を満たすと(ステップ330)、データ分割を行ってコーディングし(ステップ340)、そうでなければ次の(i+1)番目の三角形の符号化を進行させる(ステップ350)。全てのメッシュが全て符号化されるまでかかる過程

32

現在復元されたccはパーティション内の最後のc

現在復元されたccはパーティション内の最後のc

を繰返して行うことによってパーティション化がなされる。

【0069】ここで、考慮すべき点はメインブランチ上に存する分岐三角形における分割決定方法である。即ち、基本的な分割位置はY-頂点の問題を考慮するためにメインブランチ上でのみ定義されるという仮定の下で、方向性情報により分岐三角形に続いて符号化されると選択された従属ツリーを、分岐三角形が含まれたバケット内に含めて符号化するか否かを決定すべきである。これを決定するために本発明では、符号化する従属ツリーを予め符号化する方法を使用する。即ち、予め符号化した時、従属ツリーの符号化中にそのバケットが符号化されたデータで完全に満ちた場合、メインブランチから分割するという基本条件を違背するために、図18(C)のように以前の分岐三角形で分割を行い、従属ツリーまで符号化された総ビット発生量がバケットの大きさより小さい場合には、図18(A)または図18(B)のような形態で従属ツリーをバケットに含めて符号化する。

【0070】第2方法は、バケット許容値に到達しなかったが、符号化する次の三角形が分岐三角形であり、分岐する従属ツリーから発生すると予測されるビット量と現在のパーティションから発生したビット量との和がバケットの大きさを越えると判断されれば分割を行う。これを詳しく説明すれば次の通りである。

1. 各連結成分の最初のパーティションで符号化されたビット数(cur<sub>bits</sub>)を貯蔵する。  
【0071】2. ターゲットビット(target bit)に到達しなかったが、次の三角形が分岐三角形ならば、次の三角形における従属ツリーの大きさ(n<sub>st</sub>)を求める。3. 現在のパーティションで今まで符号化された三角形の頂点数(cur<sub>ng</sub>)を求める。  
4. cur<sub>ng</sub>+n<sub>st</sub>+2>n<sub>q</sub>であれば分割するが、次の通り分割を行う。

【0072】(1) 従属ツリーから発生するビット予測量がバケットより大きければ次の分岐三角形までのみ含んで分割をする。即ち、n<sub>st</sub>+2>n<sub>q</sub>なら図18(C)のように次の分岐三角形のみを含んで分割し、(2)そうでなければ現在の三角形で分割する。第3方法は、バケット許容値を設定せず、常に次に符号化するメッシュの1つのバケット単位を予め符号化し、バケットの大きさを満たす時の三角形の総数を計算した後、この三角形の個数情報を用いて実際のバケット単位のビットストリームを構成する方法である。この方法でも前記方法で考慮した従属ツリーをバケット内に含むか否かの決定方法を同一に適用する。

【0073】一方、前述したような方法でメッシュデー

(18)

特開 2000-194843

33

タをパーティション化する時さらに考慮すべき2つの事項がある。第1、Y-頂点の計算に伴う復号化時の遅延時間の発生問題であり、第2、分割されたデータ間の独立性保障問題である。本発明では、第1問題に対しては前述したように仮想連結成分を設定する方法により遅延時間を最小化させ、第2問題に対しては後述するように方向性(orientation)情報、バウンディングルーブインデックス(vi)情報と三角形エッジ(polygonedge)情報をバケット単位で定義して解決する。

【0074】<仮想連結成分を用いたY-頂点処理方法> tt/tc対の分割されたデータ間の相互独立性を保ちながらY-頂点を処理するために2つの方法が用いられる。第1方法は最も簡単な方法であって分割されたデータの長さとY-頂点情報を復号化器に送ることであり、第2方法はtt/tc対の特性に応じて分割したデータを仮想連結成分として定義する方法である。第1方法によれば、全体メッシュの長さと分割されたメッシュの長さとが、復号化時に分かるので、復元されるメッシュの構造が推定でき、さらに、Y-頂点の情報も分かって漸進的なレンダリングが行える。しかし、これら付加情報によって符号化

効率が劣る短所がある。

【0075】第2方法の仮想連結成分を定義する方法において、2進ツリーで構成されるtt/tc対は、式③のようにブランチとリーフの個数で連結成分を定義するが、分割されたメッシュデータに式③を満たすように仮想ビットを定義すれば仮想連結成分を構成しうる。この際、パーティションの開始位置は、常に2進ツリーのメインブランチのうちランでのみ発生するように構成する。これはY-頂点に対する処理を容易にするための条件であって、例えばメインブランチでない所で分割を行うと、前述したY-頂点の問題が発生し、Y-頂点の処理のための追加情報が提供されるべき、或いは復元及びレンダリングをパーティション単位別に行えない問題が発生する。

【0076】ブランチ数+1=リーフの数…③

仮想ビットを定義する方法は2つの場合に分けて定義しうる。

1. ランやリーフでパーティションが終わる場合

式③を満たすために(trun、tleaf)に(1、1)の1つの仮想ビット対を追加する。

【0077】2. ブランチでパーティションが終わる場合

式③を満たすために(trun、tleaf)に(1、1)の二つの仮想ビット対を追加する。さらに、ブランチでパーティションが終わり、頂点毎に(per vertex)幾何、色相、法線、テクスチャー座標情報を処理する場合、分岐三角形の最後の三角形に対するこれらの情報は符号化しない。

これは、現在のパーティションの分岐直前の三角形と、\*

right\_index-left\_index-1現在のパーティションのtt情報のみを復元した

時発生した三角形の総数…④

2. ブランチにおける仮想ビットの有無判別

34

\* 次に符号化するパーティションのルート三角形のインデックス情報により決定されるY-頂点インデックスとにより分岐三角形の復元ができるからである。但し、次に符号化するパーティションに損失が発生する場合、現在のパーティションの最後に現れる分岐三角形は復元できないとの短所がある。前記問題は分岐三角形の両側ブランチから派生される二つの従属ツリーのうち、任意の1つの従属ツリーの総大きさ情報(lb1)を、tt/tc対の情報に含めて復号化器に伝送することで解決しうる。これはlb1値を用いて、次に復元するパーティション情報無しで分岐三角形のY-頂点インデックスが計算できるからである。一方、フェース毎またはコーナー毎に符号化する場合には、Y-頂点インデックスに依存しないために、全ての場合に対して符号化が行われる。図18(A)乃至図18(C)は仮想連結成分を構成する概念図である。図18(A)はリーフでパーティションが終わる場合、図18(B)はランでパーティションが終わる場合、そして図18(C)はブランチでパーティションが終わる場合の仮想三角形を示しており、▲表示の仮想三角形は仮想ビットで表される。

【0078】一方、図18(A)において、分岐三角形以降に直ちにリーフ三角形が出た形態で分割されると、図18(C)のようなツリー構造となる。この場合、復号器では図18(A)と図18(C)の形態を区分できなくなるため、符号化部ではツリーブランチにおける端部を除いては、分岐三角形の直後のリーフ三角形で分割が発生しないようにすべきである。こうして復号部では、図18(A)と図18(C)の形態を区分するための他の情報無しで全ての三角形に対する復号化が可能となる。

【0079】従って、メインブランチでのみ分割するという基本条件下では、従属ツリーが大きな場合にはバケット化できない問題があったが、前記方法のように仮想連結成分を構成する方法を使用すれば、従属ツリーを多数個の相互独立した仮想連結成分で構成でき、全ての場合に対してバケット化しうるという付加的な長所が発生する。

【0080】さらに、かかる方法で仮想連結成分を構成して符号化した場合、復号化器で仮想ビットを判断する方法は次の通りである。

1. ランとリーフにおける仮想ビットの有無判別

式④を満たすと仮想ビットが存在し、そうでなければ存在しない。図18(A)と図18(B)のように、符号化された三角形のうち最後から3番目の三角形が分岐三角形でなく式④を満たすと、(trun、tleaf)で(1、1)対の仮想情報のみ発生したと判断し、仮想三角形に対してはtdデータを復号化しない。

【0081】

50 図18(C)のように分岐三角形から分割された場合、仮想

リーフ三角形が二つ加えられる。従って、ttデータの最後から三番目の三角形が分岐三角形であり式④を満たすと、(trun、tleaf)で(1、1)の二対の仮想ビット情報が存在するので、最後の2つのリーフ三角形に対してはtdデータは復号化しない。

【0082】<多角形メッシュにおけるデータ分割>位相幾何学的サーザリでは、多角形で構成されたメッシュ情報をコーディングするために、まず多角形情報を三角形に再構成する。図19は、この例を示すが、実線は元多角形メッシュの実際エッジ、点線は多角形を三角形に分割するために加えられた仮想エッジを示す。多角形を三角形に分割するために仮想エッジが加えられるが、これを復号化器で元の多角形に復元するためには、仮想エッジを除去するための情報が復号化器に伝送されるべきである。この情報を多角形エッジ(polygon edge)情報と称する。1つの三角形当り1つの多角形エッジ情報が伝送される場合、1なら実際エッジを、0なら仮想エッジを表す。

【0083】従来の方法では、メッシュに含まれた多角形を三角形メッシュ化した後、最初の三角形を除いた全ての三角形の数だけ多角形エッジ情報を発生させる。ここで、符号化器は最初の三角形を実際エッジとして設定するので、復号化器では最初の三角形の多角形エッジ情報を1と仮定して復元させる。しかし、図20(B)のように、三角形でない多角形の内部でパーティション化のためにデータを分割させるべき場合、従来の方法下ではメッシュ情報が復元できない問題が発生する。また、これによってバケットの大きさを満たすようにメッシュデータを分割するに当たって、制約となって符号化効率を低下させる要因となる。

【0084】従って、1つのパーティションが仮想エッジ上で始まる場合には、パーティション内に定義される多角形エッジ情報をパーティションの最初の三角形に対しても定義しなければ、前記制約及び短所が無くせない。図20(D)及び図20(E)でttは三角形ラン情報、tmlはマーチ情報、peは多角形エッジの情報、下添字はttに該当する同一順序におけるtmとpeを、nlは三角形の個数を意味する。

【0085】図20(D)は多角形メッシュを実際エッジ上でのみ分割する場合の文法であって、図20(C)が対応され、図20(E)は多角形メッシュを仮想エッジ上で分割することが許容される場合の文法であって、図20(B)の場合が対応される。従って、本発明では実際エッジで分割することを基本とするが、場合に応じて多角形の内部で分割されても復元可能にする文法を提供する。これに各パーティションにおける多角形エッジの処理は、次のようにパーティションの形態情報ptを用いたり、メッシュ内の多角形の存否情報(triangulated)とpolygon\_edge情報を用いる方法で定義しうる。

【0086】1. パーティションの分割された形態情報

(19) 特開2000-194843

35

ptに応じる処理方法

(1) 第0パーティション類型(pt=0)の場合、最初の多角形エッジ値はコーディングしない。

(2) 第2パーティション類型(pt=2)であり、パーティション内のメッシュに1つ以上の多角形が存在する場合(triangulated=0)にのみ最初の多角形エッジ値をコーディングする。

【0087】2. triangulatedとpolygon\_edge情報に応じる処理方法

(1) 実際エッジ上で分割する場合パーティション内の最初の多角形エッジ値はコーディングしない。

(2) 仮想エッジ上で分割を許す場合パーティション内の最初の多角形エッジ値をコーディングする。

【0088】ここで、任意の多角形で実際エッジと仮想エッジとの判別は次のような条件式によって定義される。

if(triangulated=1)実際エッジ  
else if(polygon\_edge=0)仮想エッジ  
else 実際エッジ

20 従って、復号化器ではtriangulatedが0の場合、最初の多角形エッジ値(polygon\_edge)を無条件復元してその値が0なら最初の多角形エッジ値として、1なら2番目の多角形エッジ値として指定する。

【0089】triangulatedという情報は1つのパーティション内のメッシュに多角形が1つ以上存在すれば0、そうでなければ1を表示してパーティション単位で1ビット情報を構成する。

<方向性情報によるデータ分割方法>前述した方向性情報の必要性及び定義方法はデータを分割する場合にも同一に適用される。但し、ここでさらに考慮すべき事項は、前述した仮想連結成分を用いる場合、従属ツリーにおいても分割しうるために、以前パーティションに損失が発生して復元されない場合、復元する現在のパーティションのメッシュがメインブランチに連結されたものか、サブブランチに連結されたものかが判断できないという点である。従って、バウンディンググループインデックスの計算エラーが誘発され、復元及びレンダリングがなされない問題が発生しうる。

【0090】従って、本発明では、方向性情報を実際メッシュの連結成分内で定義する方法と同一な方法で仮想連結成分内でも全て定義することによって仮想連結成分内で独立して復元及びレンダリング可能にする。

<バウンディンググループ情報(vi)を含むデータ分割方法>バウンディンググループ情報には実際頂点の幾何情報に対するインデックス値がマッピングされており、バウンディンググループのインデックスがわかるだけでマッピングされた三角形の頂点の実際座標値がわかる。従って、損失が発生したパーティションの再伝送がない場合、図9のように、次に復元するパーティションは、パーティションの開始位置に存する三角形に対するバウンディン

50

37

グループインデックス情報がわからないので復元できない。このような非効率性を防止するためにはそれぞれのパーティションが独立して復元及びレンダリング可能にすべきである。このためには各パーティション内の最初の三角形が始まるバウンディンググループ上における開始位置値を必ず指定すべきである。

【0091】復号化器でメッシュを復元する過程中、vqは次に復元する三角形の各頂点値をバウンディンググループというテーブルに貯蔵させるのに使われる。この際、各三角形の頂点のインデックス値は、図21のように、開始位置にある三角形の頂点のインデックス値を1だけ増減した値でテーブルに貯蔵される。従って、パーティションの最初の三角形の頂点のバウンディンググループにおけるインデックスのみ決定できるなら、次に復元する残り三角形の頂点は最初の三角形の頂点のバウンディンググループインデックスから1だけ増減させて決定しうる。

【0092】従って、本発明ではパーティションの最初の三角形の頂点のバウンディンググループインデックスのみを各該当パーティション毎に指定することによって、それぞれのパーティションに対する独立性を保障可能にする。さらに、本発明で定義したviは、図9のように、左側インデックス及び右側インデックスの2種のインデックス情報で構成される。これに対するビットストリームの構成は図22のようである。図22においてL(left index)は三角形ストリップの左側境界上の最初の頂点のバウンディンググループにおけるインデックス、R(right index)は右側境界上の最初の頂点のバウンディンググループにおけるインデックスを意味する。

【0093】一方、ビットストリームがCDのような貯蔵媒体を通じて順次に受信されたり、復元される場合にバウンディンググループのインデックスは連結成分(cc)別に0から始めてバウンディンググループの大きさの間に存在するので、各パーティションのヘッダ部分に与えるインデックス値もその間の値で定義すればよい。これはバウンディンググループとtt/td対情報とが正確に一对一の対応関係であり、常にvqに続いてtt/td対の順に符号化されるために可能となる。しかし、伝送媒体の特性に応じては、伝送遅延による送信順序及び受信順序が異なる場合が発生したり、ビットストリームの損失が発生することもある。こういう場合、tt/td対と対応されるvq情報が順次に受信または復元されると保障できないために、この場合に限ってはインデックス値を他の方式で指定する必要がある。図21は、多角形メッシュが多数個の連結成分を有する場合のバウンディンググループとtt/td対との関係を示す。ここで、最初の列は連結成分別にバウンディンググループに独立してインデクシングする場合、2番目の列は現在のバウンディンググループのインデクシングを以前バウンディンググループの最後の値に連続して増加させる場合を示す。図23(A)及び図23(B)は、この2つの場合に対してインデックス情報を指定する方法の差

(20)

特開2000-194843

38

を示す。例えば、最初の連結成分のバウンディンググループの大きさがn1、2番目の連結成分のバウンディンググループの大きさがn2の場合、2番目の連結成分のパーティションのためのインデックス値は、図23(A)及び図23(B)のように、2つの方式でインデックス情報がヘッダに与えられる。図23(A)は連結成分別バウンディンググループインデクシングを行う場合のパーティションのヘッダ情報を示し、図23(B)はメッシュ全体に亘ってバウンディンググループインデクシングを行う場合のパーティションのヘッダ情報を示す。

【0094】<幾何情報を考慮したデータ分割方法>今まではデータパーティションを、モデルの連結情報を中心に本発明を説明した。ここでは幾何情報のパーティション間相互独立性保障と符号化効率を向上させる方法に対して記述する。データの分割時、各三角形の頂点が以前パーティションに含まれた三角形の頂点と接している場合、その幾何情報が既に符号化されたのか否かに関する情報が必要である。これはvisited表示子により定義されるが、visitedが1なら既に符号化された場合を、0なら符号化されていない場合を示す。一般に、以前パーティションと現在のパーティションの両方ともに用いられる幾何情報は2つのパーティションの境界から現れ、現在のパーティションを符号化する場合、以前パーティションの境界に位置した幾何情報は全てvisited=2で定義されている。このような点を考慮すれば次のような符号化方法が可能である。

【0095】1. 図24(A)のように、現在のパーティションでは以前パーティションにより訪問されない幾何情報に対してのみ符号化する方法。

2. 図24(B)のように、以前の他のパーティションでコーディングされた情報を、現在のパーティションで重畳して符号化することによって、以前パーティションと独立して現在のパーティションのみでも幾何情報を復元可能にする方法。

【0096】3. 図24(C)のように、以前の他のパーティションで既に符号化された幾何情報のうち、現在のパーティションで新たに現れる多数個の幾何情報と連結されて現れる重要な幾何情報に対してのみ重畳して符号化する方法。

4. 図24(D)のように、以前パーティションと現在のパーティションに重畳されて現れる幾何情報は、一般に三角形ストリップの一方の境界面で連続されて現れるので、重畳されるデータをサンプリングして重畳されるデータの半分は以前パーティションで、残り半分は現在のパーティションで符号化する方法。

【0097】図24(A)乃至図24(D)において、陰影円は既に訪問された幾何情報、黒円は以前パーティションと現在のパーティションで重畳符号化される幾何情報であり、白円はまだ訪問されていない幾何情報を意味し、太線はパーティションの境界を示すものである。このよう

(21)

特開2000-194843

39

な方法は各々次のような長短所を有している。

【0098】▲1の方法は具現が容易で符号化効率が良いが、予測する周辺幾何情報が他の方法に比べて相対的に少ないため正確度が劣る恐れがある。

▲2の方法は具現が容易で周辺の全ての幾何情報が参照できて正確度は相対的に高いが、重畳して符号化することによって符号化効率面では最も劣る。

▲3の方法は圧縮率を保ち、幾何情報の損失も小さいが、境界面の周辺における連結情報の特性を予め把握すべきなので具現しにくく、複雑度も増加する。

【0099】▲4の方法は2の方法の短所は補完できるが、レンダリングにおいて遅延時間が発生する。これは1、2、3の方法はそれ自体のみで即時に復元及びレンダリング可能であるが、4の方法は略された値を周辺の幾何情報を用いて補間するか、それとも次のパーティションが復元されてからこそレンダリングしうるからである。

【0100】一方、1、2の方法は、実際にデータを分割する時、各パーティション毎に適応的に適用させうる。即ち、符号化効率及びパーティション間相互独立性の保障を考慮して、データ分割の開始と終了位置がメインランチにあるパーティションは、以前に符号化されたパーティションから訪問情報が正確に分かるために1の方法で、データ分割の開始と終了位置が従属ツリーにあるパーティションは、以前パーティションに損失が発生して復元できなかった場合、訪問情報がわからないために2の方法を適用してパーティション化しうる。本発明で \*

```

if(a,b,cを全て利用できない場合)d'=0
else if(a,b,cのうち1つのみ利用できる場合)d'=t
else if(a,b,cのうち2つのみ利用できる場合)[
  if(頂点二つのそれぞれの距離がdに対して1の場合)d'=(t1+t2)/2
  else if(1つの頂点(=t1)距離のみdに対して1の場合)d'=t1
  else
    d'=t2
]else
  d'=f(a,b,c) ...⑥

```

ここでtはa、b、cの三つの頂点のうち利用可能な任意の1つの頂点を、t1とt2はa、b、cのうち利用可能な二つの頂点を意味する。

【0104】＜幾何情報及び画像情報のビットストリーム構成及び処理方法＞幾何情報と色相情報、法線情報及びテクスチャ座標などの画像情報を符号化する方式は次の通りである。まず、図25(A)のように、1つの三角形のマーチビットが現れる度に関連した特性情報を符号化する方式であって、復号化器でマーチビットと多角形エッジ情報を各々1つずつ復元すれば、直ちに三角形のレンダリングが可能になる特徴がある。他の方法は1つのデータパーティション内に現れる全ての特性情報を、図25(B)のように情報特性別に分離して符号化する方法である。

【0105】図26はかかる文法下で実際に画像情報を符号化するための流れ図である。図26によれば、幾何情報

40

\*は各、パーティションのヘッダ情報に前述した幾何情報符号化方法に関する情報を提供するが、前記1と2の方法のみを適応的に適用する場合、1ビットの境界予測(boundary prediction)表示子bpを用いて0なら1の方法で、1なら2の方法で幾何情報が符号化されたパーティションであることを表示する。図24(E)は境界予測表示子がヘッダ情報に含まれた構造の文法を示す。

【0101】＜位置情報の予測符号化方法＞メッシュの予測符号化方法は、任意の1つの頂点の位置情報(=d)を、既に符号化された隣接した三角形の三つの頂点(a、b、c)の位置情報を用いて予測(=d')し、予測値と実際値との差を符号化して符号化効率を高めることを目的とする。このような予測方法は式⑤のように定義される。

【0102】 $d' = f(a, b, c) \dots ⑤$

一方、このような位置情報の予測方法はbp値によってその方法を変えるべきである。これは予測に用いられる隣接した三角形の三つの頂点を利用するか否かが境界予測の方法に応じて変わるために、パーティション単位の独立的復元を全ての方法に対して保障できないからである。従って、本発明ではbp値が1の場合、即ち重畳が許容される場合には、周辺隣接頂点のvisited情報のみを用いて既存の方法と同一に予測符号化方法を適用させ、bp値が0の場合、即ち重畳が許容されず自体パーティション内の位置情報のみで予測符号化を行うべき場合には式⑥のように予測符号化を行わせる方法を提示する。

【0103】

符号化方法としては図24(A)と図24(B)に示された方式を混用し、図25(A)の文法体系を用いる方式を示したものである。図26は、多角形3次元モデルをパーティション単位で分割して符号化する全体的な順序図である。

【0106】1. 境界予測方法を決定する(401段階)。即ち、図24(A)方式で符号化するか、図24(B)の方式で符号化するかを決定する。

2. ルート三角形(パーティションにおいて最初に現れる三角形)を符号化する(402段階)。

3. ルート幾何情報を符号化する(403段階)。

【0107】4. 次の三角形が存在すれば(404段階)、次の三角形に移動した後(405段階)、次の頂点に移動する(406段階)。

5. 三角形の各頂点に対して以前パーティションで訪問されたのかを判断し(407段階)、符号化されなかったら現在のパーティションで符号化されたのかを判断し(409

(22)

特開 2 0 0 0 - 1 9 4 8 4 3

41

42

段階)、符号化されなかったらこれを符号化する(410段階)。

【0108】6. 頂点が以前パーティションで符号化されたり、或いはbp値が1なら(408段階)、この値が現在のパーティションで既に符号化されたかを判断し(409段階)、符号化されていないと符号化する(410段階)。

7. 前記405段階乃至410段階の過程を最後の三角形を符号化するまで反復処理する。

【0109】以下、本発明に係る漸進的な三次元メッシュ情報及び損失に対する弾力性を具現するための符号化文法の一例を示す。3次元メッシュに対する圧縮されたビットストリームは、図27に示すように、グローバル情報を有するヘッダデータブロックとこれに従って各々3次元メッシュの一つの連結成分に関連した一連の連結成分データブロックで構成されている。

【0110】もし、3次元メッシュがエラー回復的なモードで符号化されると、連結成分データブロックは図28に示すように、パーティションにグループ化または分割される。各連結成分データブロックは図29に示すように、頂点グラフレコード、三角形ツリーレコード及び三角形データレコードの3つのレコードで構成される。

【0111】三角形ツリーレコードはsimple多角形を形成する、対応する連結成分の全ての三角形をリンクする三角形スパニンググラフの構造を有する。多角形メッシュはビットストリームにおいて三角形に分割された形態で表現されるが、それは本来のフェースを再生するのに必要な情報も有する。頂点グラフレコードは現在の連結成分だけでなく、以前に復号化された連結成分内で本来の連結情報を再生するためのsimple多角形の境界エッジの対をステッチするのに必要な情報を有する。その連結情報は(連結成分当りの)グローバル情報と(三角形当りの)ローカル情報とに分割される。グローバル情報は頂点グラフと三角形ツリーレコードに貯蔵される。ローカル情報は三角形データレコードに貯蔵される。三角形データは図30に示すように、三角形単位に基づいて整列されるが、三角形の順序は三角形ツリーの訪問(traverse)により決定される。

【0112】与えられた三角形に対するデータは図31＊

パーティションタイプの定義

3D_MOBL_start_code	パーティションタイプ	意味
0000 0000 0011 0001	partition_type_0	vg, tt及びtdの1つ以上の「A-B」
0000 0000 0011 0011	partition_type_1	1つ以上のvg
0000 0000 0011 0100	partition_type_2	1つのtt及びtd対

mobl\_id: この8ビットの符号無し整数はメッシュ客成分(mesh object component)に対する唯一の識別子を示す。

one\_bit: このブール値(boolean value) は常に真である。

【0118】last\_component: このブール値はさらに復号化する連結成分の存否を示す。もし、last\_compon

＊のように構成される。マーチパターン、td\_orientation及びpolygon\_edgeは三角形単位の連結情報を構成する。他のフィールドは頂点座標(coord)及び選択的に法線(normal)、色相(color)及びテクスチャー座標(texCoord)情報を再生するための情報を有する。

【0113】以下、図32～図72の符号化文法の一例に含まれた各項目に対して具体的に説明する。

▲3D\_Mesh\_Object

3D\_MO\_start\_code: これは同期の目的として用いられる長さ16ビットの唯一のコードである。このコードの値は常に'0000 0000 0010 0000'である。

▲3D\_Mesh\_Object\_Layer

3D\_MOL\_start\_code: これは同期の目的として用いられる長さ16ビットの唯一のコードである。このコードの値は常に'0000 0000 0011 0000'である。

【0114】mol\_id: この8ビットの符号無しの整数は段階別メッシュ(mesh object layer: MOL)に対する唯一の識別子を示す。値0はベース段階(base layer)を示し、0より大きな値は精錬段階(refinement layer)を示す。3D\_Mesh\_Object\_Header後の最初の3D\_Mesh\_Object\_Layerはmol\_id=0、同一な3D\_Mesh\_Objectを有する以降の3D\_Mesh\_Object\_Layerはmol\_id>0になるべきである。

【0115】cqd\_n\_verticesは3次元メッシュの現在の解像度における頂点の個数である。計算の簡略化のために使われる。cqd\_n\_trianglesは3次元メッシュの現在の解像度における三角形の個数である。計算の簡略化のために使われる。cqd\_n\_edgesは3次元メッシュの現在の解像度におけるエッジの個数である。計算の簡略化のために使われる。

【0116】▲3D\_Mesh\_Object\_Base\_Layer

3D\_MOBL\_start\_code: これは同期の目的として用いられる長さ16ビットの唯一のコードである。またエラー弾力性のために用いられるパーティションの3つの他類型を示すために使われる。

【0117】

【表2】

entが真であれば、最後の成分は復号化された。そうでなければ、復号化する成分がさらに存在する。このフィールドは算術符号化される。

codap\_last\_vq: このブール値は現在のvqがパーティションにおける最後のvqなのかを示す。パーティションに復号化されるべきvqがさらに存在すれば、偽となる。

【0119】codap\_vq\_id: この符号無し整数はtt/td

43

対が使用すべき頂点グラフの識別子を示す。この値の長さのlog\_vqid\_lenは以前のpartition\_type\_1から復号化されたvgのvg\_numberのログスケール値である。もし、以前のpartition\_type\_1に一つのvgのみが存在すれば、codap\_vq\_idは符号化されない。  
codap\_left\_bloop\_idx: この符号無し整数はパーティションから三角形ストリップを再生するための境界ループテーブルにおける左側開始インデックスを示す。この値の長さのlog\_bloop\_lenは境界ループテーブルの大きさ値をログスケールした値である。  
【0120】codap\_right\_bloop\_idx: この符号無し整数はパーティションから三角形ストリップを再生するための境界ループテーブルにおける右側開始インデックスを示す。この値の長さのlog\_bloop\_lenは境界ループテーブルの大きさ値をログスケールした値である。  
codap\_bdry\_pred: このブールフラグは二つ以上のパーティションにおいて共通の幾何情報及び画像情報をどのように予測するかを示す。もし、codap\_bdry\_predが'1'ならば全ての共通情報が現在のパーティションで予測され、そうでなければ共通情報はただ1つのパーティションでのみ予測される。  
【0121】▲3D\_Mesh\_Object\_Header  
ccw: このブール値は復号化されるフェースの頂点順序が逆時計回り方向の順序を従うか否かを示す。  
convex: このブール値モデルが凸状なのか否かを示す。\*

44

\* solid: このブール値はモデルが堅固なのか否かを示す。  
【0122】creaseAngle: この6ビットの符号無し整数はクリース角(crease angle)なのか否かを示す。  
▲coord\_header  
coord\_binding: この2ビットの符号無し整数は3次元メッシュに対する頂点座標の結合を示す。唯一に許容される値は'01'である。  
【0123】coord\_bbox: このブール値は幾何情報に対してバウンディングボックス(bounding box)が提供されるか否かを示す。バウンディングボックスが提供されないと、デフォルトが使われる。  
coord\_xmin, coord\_ymin, coord\_zmin: この浮動小数点値は幾何情報が置かれたバウンディングボックスの左下コーナーを示す。  
【0124】coord\_size: この浮動小数点値はバウンディングボックスの大きさを示す。  
coord\_quant: この5ビットの符号無し整数は幾何情報に対する量子化ステップ(quantization step)を示す。  
coord\_pred\_type: この2ビットの符号無し整数はメッシュの頂点座標を再生するために用いられる予測の形態を示す。  
【0125】  
【表3】

coord\_nlambda: この2ビットの符号無し整数は幾何情報を予測するために用いられる先祖(ancestor)の個数を示す。coord\_nlambdaに対して許容可能な値は3である。表4はcoord\_pred\_typeの機能として許容可能な値※

※を示す。  
【0126】  
【表4】

coord_pred_typeの機能としてcoord_nlambdaに対して許容可能な値	
coord_pred_type	coord_nlambda
00	not coded
01	3

coord\_lambda: この符号無し整数は予測のための先祖に与えられる加重値を示す。このフィールドにおいて用いられるビット数はcoord\_quant+3と同じである。  
【0127】▲normal\_header

40★メッシュに対するノーマルの結合を示す。許容可能な値は表5に記述される。  
【0128】  
【表5】

normal\_binding: この2ビットの符号無し整数は3次元 ★  
normal\_bindingに対して許容可能な値

normal_binding	binding
00	not bound
01	bound per vertex
10	bound per face
11	bound per corner



(24)

特開 2000-194843

45

46

normal\_\_bbox: このブール値は常に偽('0')でなければならない。

normal\_\_quant: この5ビットの符号無し整数はノーマルに対して用いられる量子化ステップを示す。

\* 【0 1 2 9】 normal\_\_pred\_\_type: この2ビットの符号無し整数はどのように法線値が予測されるかを示す。

【0 1 3 0】

\* 【表 6】

normal\_pred\_typeに対して許容可能な値

normal_pred_type	予測類型
00	no prediction
01	tree prediction
10	parallelogram prediction
11	reserved

【0 1 3 1】

※ ※ 【表 7】

normal\_bindingとnormal\_pred\_typeの許容可能な結合

normal_binding	normal_pred_type
not bound	not coded
bound per vertex	no prediction, parallelogram prediction
bound per face	no prediction, tree prediction
bound per corner	no prediction, tree prediction

normal\_\_nlambda: この2ビットの符号無し整数はノーマルを予測するために用いられる先祖の個数を示す。

normal\_\_nlambdaに対して許容可能な値は1、2、3である。

normal\_\_nlambdaに対して許容可能な値を示す。

normal\_\_pred\_\_typeの機能として許容可能な値を示す。

normal\_predictionの機能としてnormal\_nlambdaに対する許容可能な値

normal_pred_type	normal_nlambda
no prediction	not coded
tree prediction	1, 2, 3
parallelogram prediction	3

normal\_\_lambda: この符号無し整数は予測のための先祖に与えられる加重値を示す。

normal\_\_lambda: このフィールドにおいて用いられるビット数はnormal\_\_quant+3と同じである。

【0 1 3 3】 ▲ color\_\_header

color\_\_binding: この2ビットの符号無し整数は3次元メッシュ

color\_bindingに対して許容可能な値

color_binding	binding
00	not bound
01	bound per vertex
10	bound per face
11	bound per corner

color\_\_bbox: このブール値は色相に対してバウンディングボックスが与えられるか否かを示す。

color\_\_rmin, color\_\_gmin, color\_\_bmin: この浮動小数点値はRGB空間内のバウンディングボックスの左下側コーナーの位置を示す。

【0 1 3 5】 color\_\_size: この浮動小数点値は色相バウンディングボックスの大きさを示す。

☆ ッッシュに対する色相の結合を示す。許容可能な値は表9に記述される。

30 【0 1 3 4】

【表 9】

40 color\_\_quant: この5ビットの符号無し整数は色相に対して用いられる量子化ステップを示す。

color\_\_pred\_\_type: この2ビットの符号無し整数はどのように色相が予測されるかを示す。

【0 1 3 6】

【表 10】

(25)

特開2000-194843

47

48

normal\_pred\_typeに対して許容可能な値

color_pred_type	予測類型
00	no prediction
01	tree prediction
10	parallelogram prediction
11	reserved

【0137】

\* \* 【表11】

color\_bindingとcolor\_pred\_typeの許容可能な結合

color_binding	color_pred_type
not bound	not coded
bound per vertex	no prediction, parallelogram prediction
bound per face	no prediction, tree prediction
bound per corner	no prediction, tree prediction

color\_nlambda: この2ビットの符号無し整数は色相を ※ color\_pred\_typeの機能として許容可能な値を示す。

予測するために用いられる先祖の個数を示す。color\_n 【0138】

lambdaに対して許容可能な値は1、2、3である。表12はc※ 【表12】

color\_predictionの機能としてcolor\_nlambdaに対する許容可能な値

color_pred_type	color_nlambda
no prediction	not coded
tree prediction	1,2,3
parallelogram prediction	3

color\_lambda: この符号無し整数は予測のための先祖  
に与えられる加重値を示す。このフィールドにおいて用  
いられるビット数はcolor\_quant+3と同じである。

【0139】 ▲texCoord\_header

★元メッシュに対するテクスチャの結合を示す。許容可  
能な値は表13に記述される。

【0140】

【表13】

texCoord\_binding: この2ビットの符号無し整数は3次 ★

texCoord\_bindingに対して許容可能な値

texCoord_binding	Binding
00	not bound
01	bound per vertex
10	forbidden
11	bound per corner

texCoord\_bbox: このブール値はテクスチャに対して  
バウンディングボックスが与えられるか否かを示す。texCoord\_umin, texCoord\_vmin: この浮動小数点値は  
2次元空間内のバウンディングボックスの左下側コー  
ナーの位置を示す。

【0141】 texCoord\_size: この浮動小数点値はテク

スチャバウンディングボックスの大きさを示す。 ☆40

☆texCoord\_quant: この5ビットの符号無し整数はテクス  
チャに対して用いられる量子化ステップを示す。texCoord\_pred\_type: この2ビットの符号無し整数は  
どのように色相が予測されるかを示す。

【0142】

【表14】

texCoord\_pred\_typeに対して許容可能な値

texCoord_pred_type	予測類型
00	no prediction
01	forbidden
10	parallelogram prediction
11	reserved

【0143】

【表15】

(26)

特開 2 0 0 0 - 1 9 4 8 4 3

49

50

texCoord\_bindingとtexCoord\_pred\_typeの許容可能な結合

texCoord_binding	texCoord_pred_type
not bound	not coded
bound per vertex	no prediction,parallelogram prediction
bound per corner	no prediction,tree prediction

texCoord\_nlambda: この2ビットの符号無し整数はテクスチャーを予測するために用いられる先祖の個数を示す。texCoord\_nlambdaに対して許容可能な値は1、2、3である。表16はtexCoord\_pred\_typeの機能として許容可能な値

\* 可能な値を示す。  
【 0 1 4 4 】  
【 表 1 6 】

texCoord\_predictionの機能としてtexCoord\_nlambdaに対する許容可能な値

texCoord_pred_type	texCoord_nlambda
no prediction	not coded
tree prediction	1,2,3
parallelogram prediction	3

texCoord\_lambda: この符号無し整数は予測のための先祖に与えられる加重値を示す。このフィールドにおいて用いられるビット数はtexCoord\_quant+3と同じである。

【 0 1 4 5 】 ▲Cqd\_header

cqd\_n\_proj\_surface\_spheresは投射された表面球(Projected Surface Spheres)の個数である。典型的に、この個数は1と同一である。cqd\_x\_coord\_center\_pointは投射された表面球の中心点(典型的に、客体の重さ中心点)のx座標である。

【 0 1 4 6 】 cqd\_y\_coord\_center\_pointは投射された表面球の中心点のy座標である。cqd\_z\_coord\_center\_pointは投射された表面球の中心点のz座標である。cqd\_normalized\_screen\_distance\_factorは投射された表面球の半径と比較して仮想スクリーンがどこに位置するかを示す。投射された表面球の中心点と仮想スクリーンとの距離はcqd\_radius/(cqd\_normalized\_screen\_distance\_factor+1)と同一である。cqd\_radiusは各投射された表面球に対して記述されるが、cqd\_normalized\_screen\_distance\_factorは一度だけ記述される。

【 0 1 4 7 】 cqd\_radiusは投射された表面球の半径である。cqd\_min\_proj\_surfaceは対応する投射された表面球に関する最小の投射された表面値である。この値は度々cqd\_proj\_surface値のうち1つと同一である(しかし、必ずしもそういうことではない)。cqd\_n\_proj\_pointsは投射された表面が伝送される投射された表面球上の点の個数である。他の全ての点に対し、投射された表面は線形補間により決定される。cqd\_n\_proj\_pointsは典型的に最初の投射された表面球に対しては小

さく(例えば、20)、さらに投射された表面球に対しては非常に小さい(例えば、3)。

【 0 1 4 8 】 cqd\_sphere\_point\_coordは8面体内の点の位置に対するインデックスである。cqd\_proj\_surfaceはcqd\_sphere\_point\_coordにより記述された点内の投射された表面である。

▲vertex graph

vq\_simple: このブール値は現在の頂点グラフがsimpleなのか否かを指定する。simple vertex graphは如何なるループも含まない。このフィールドは算術符号化される。

【 0 1 4 9 】 vq\_last: このブールフラグは現在のランが現在の分岐頂点から始める最後のランなのか否かを示す。このフィールドは各分岐頂点の最初のラン、即ちskip\_last変数が真(true)の時には符号化されない。現在の頂点ランに対してvq\_lastの値が符号化されない場合には偽(false)であると見なされる。このフィールドは算術符号化される。

【 0 1 5 0 】 vq\_forward\_run: このブールフラグは現在のランが新たなランなのか否かを示す。もし、それが新たなランでなければ、それはグラフ内のループを示す予め訪問された(traversed)ランでなければならない。このフィールドは算術符号化される。

vq\_loop\_index: この符号無し整数は現在のループが連結されたランに対するインデックスを示す。その単一(unary)の表現は算術符号化される。もし、変数openloopsがvq\_loop\_indexと同一であれば、その単一の表現から後続する'1'は取除かれる。

【 0 1 5 1 】

【 表 1 7 】

(27)

特開2000-194843

51

52

vg_loop_index	単一の表現
0	1
1	01
2	001
3	0001
4	00001
5	000001
6	0000001
...	
openloop-1	openloop-1個の0に続いて1が従う

vg\_run\_length: この符号無し整数は現在の頂点ランの長さを示す。その単一の表現は算術符号化される。 \* 【表18】

vg_run_length	単一の表現
1	1
2	01
3	001
4	0001
5	00001
6	000001
7	0000001
8	00000001
n	n-1個の0に続いて1が従う

vg\_leaf: このブールフラグは現在のランの最後の頂点がリーフ頂点なのか否かを示す。もし、それがリーフ頂点でなければ、それは分岐頂点である。このフィールドは算術符号化される。

【0153】vg\_loop: このブールフラグは現在のランのリーフがループを示し、そのグラフの分岐頂点に連結されたのか否かを示す。このフィールドは算術符号化される。

20※▲triangle\_tree

branch\_position: この整数の変数は三角形ツリーにおいて最後のブランチ位置を貯蔵するために使われる。

【0154】tt\_run\_length: この符号無し整数は現在の三角形ランの長さを示す。その単一の表現は算術符号化される。

【0155】

※ 【表19】

vg_run_length	単一の表現
1	1
2	01
3	001
4	0001
5	00001
6	000001
7	0000001
8	00000001
n	n-1個の0に続いて1が従う

tt\_leaf: このブールフラグは現在のランの最後の三角形がリーフ三角形なのか否かを示す。もし、それがリーフ三角形でなければ、それは分岐三角形である。このフィールドは算術符号化される。

【0156】triangulated: このブール値は現在の連結成分が三角形のみを含むか否かを示す。このフィールドは算術符号化される。

marching\_triangle: このブール値は三角形ツリーにおける三角形の位置により決定される。三角形がリーフまたはブランチならmarching\_triangle=0で、そうでなければmarching\_triangle=1である。

【0157】marching\_pattern: このブールフラグは三角形ランの内部のエッジのマーチ(marching)パターンを示す。0は左からの進行(march)を示し、1は右からの

進行を示す。このフィールドは算術符号化される。

40 polygon\_edge: このブールフラグは現在の三角形のベースが3次元メッシュ客体の再生時保たれるべきエッジなのか否かを示す。もし、現在の三角形のベースが保たれないと、それは捨てられる。このフィールドは算術符号化される。

【0158】codap\_branch\_len: この符号無し整数はパーティションで再生されるべき分岐三角形のY-頂点インデックス値を計算するために分岐三角形の一方のブランチに存する従属ツリーの総大きさを示す。この値の長さは境界ループテーブルの大きさ値をログスケールした値である。

▲triangle

50 td\_orientation: この一つのビットフラグは復号化器

(28)

特開 2000-194843

53

54

にブランチにおけるtt/tc対の訪問順序を知らせる。このフィールドは算術符号化される。 \* 【0159】  
\* 【表20】

td_orientation	訪問順序
0	右ブランチから
1	左ブランチから

visited: この変数は現在の頂点が訪問されたか否かを示す。 ※ 【0160】  
※ 【表21】

visited に対する値

visited	意味
0	訪問されない
1	現在のパーティションで訪問
2	以前のパーティションで訪問

no\_ancestors: このブール変数は現在の頂点の予測のために用いられる先祖がない時真となる。

coord\_bit: このブール値は幾何情報ビットの値を示す。このフィールドは算術符号化される。

【0161】 coord\_leading\_bit: このブール値はリーディング幾何情報ビットの値を示す。このフィールドは算術符号化される。

coord\_sign\_bit: このブール値は幾何情報サンプルの符号を示す。このフィールドは算術符号化される。

coord\_trailing\_bit: このブール値はトレーリング幾何情報ビットの値を示す。このフィールドは算術符号化される。

【0162】 normal\_bit: このブール値はノーマルビットの値を示す。このフィールドは算術符号化される。

normal\_leading\_bit: このブール値はリーディングノーマルビットの値を示す。このフィールドは算術符号化される。

normal\_sign\_bit: このブール値はノーマルサンプルの符号を示す。このフィールドは算術符号化される。

【0163】 normal\_trailing\_bit: このブール値はトレーリングノーマルビットの値を示す。このフィールドは算術符号化される。

color\_bit: このブール値は色相ビットの値を示す。このフィールドは算術符号化される。

color\_leading\_bit: このブール値はリーディング色相ビットの値を示す。このフィールドは算術符号化される。

【0164】 color\_sign\_bit: このブール値は色相サ★

connectivity_update	意味
00	not updated
01	fs updated
10	reserved
11	reserved

pre\_smoothing: このブール値は現在のフォレスト分割動作が総体的に頂点位置を予測するための事前平滑化段階(pre-smoothing step)を使用するか否かを示す。

★ンブルの符号を示す。このフィールドは算術符号化される。

color\_trailing\_bit: このブール値はトレーリング色相ビットの値を示す。このフィールドは算術符号化される。

20 texCoord\_bit: このブール値はテクスチャービットの値を示す。このフィールドは算術符号化される。

【0165】 texCoord\_leading\_bit: このブール値はリーディングテクスチャービットの値を示す。このフィールドは算術符号化される。

texCoord\_sign\_bit: このブール値はテクスチャーサンプルの符号を示す。このフィールドは算術符号化される。

texCoord\_trailing\_bit: このブール値はトレーリングテクスチャービットの値を示す。このフィールドは算術符号化される。

30 【0166】 ▲3DMeshObject\_Refinement\_Layer

3D\_MORI\_start\_code: これは同期の目的として用いられる長さ16ビットの唯一のコードである。このコードの値は常に'0000 0000 0011 0010'である。

mori\_id: この8ビットの符号無し整数はフォレスト(forest)分割成分に対する唯一の識別子である。

【0167】 connectivity\_update: この2バイトの変数はフォレスト分割動作がメッシュの連結情報の精錬の結果なのか否かを示す。

40 【0168】

【表22】

50 【0169】 post\_smoothing: このブール値は現在のフォレスト分割動作が量子化加工物を除去するための事後平滑化段階(post-smoothing step)を使用するか否か

を示す。

stuffing\_bit: このブール値は常に真である。

other\_update: このブール値は頂点座標に対する更新とフォレストの如何なるツリーにも伴わないフェースとコーナーに関連した特徴に対する更新がビットストリームに従うか否かを示す。

【0170】▲pre\_smoothing parameters

pre\_smoothing\_n: この整数値は事前平滑化フィルター(pre-smoothing filter)の反復回数を示す。

pre\_smoothing\_lambda: この浮動小数点値は事前平滑化フィルターの最初の媒介変数である。 10

【0171】pre\_smoothing\_mu: この浮動小数点値は事前平滑化フィルターの第2媒介変数である。

▲post\_smoothing parameters

post\_smoothing\_n: この整数値は事後平滑化フィルター(post-smoothing filter)の反復回数を示す。

【0172】post\_smoothing\_lambda: この浮動小数点値は事後平滑化フィルターの最初の媒介変数である。

post\_smoothing\_mu: この浮動小数点値は事後平滑化フィルターの第2媒介変数である。

▲fs\_pre\_update

pfs\_forest\_edge: このブール値はエッジが今まで作られたフォレストに追加されるべきであることを示す。

【0173】▲smoothing\_constraints

smooth\_with\_sharp\_edges: このブール値はデータが平滑化不連続エッジ(smoothing discontinuity edges)を表示するビットストリームに含まれたか否かを示す。もし、smooth\_with\_sharp\_edges==0なら、何れのエッジも平滑化不連続エッジとして表示されていない。もし、平滑化不連続エッジが表示されていれば、事前平滑化フィルター及び事後平滑化フィルターはこれらを考慮する。 30

【0174】smooth\_with\_fixed\_vertices: このブール値は平滑化過程中、動かないデータがビットストリームに含まれたか否かを示す。もし、smooth\_with\_fixed\_vertices==0なら、全ての頂点を動かすことが許容される。もし、固定された頂点が表示されていると、事前平滑化フィルター及び事後平滑化フィルターはこれらを考慮する。

【0175】smooth\_sharp\_edge: このブール値は対応するエッジが平滑化不連続エッジで表示されているか否かを示す。 40

smooth\_fixed\_vertex: このブール値は対応する頂点が固定された頂点なのか否かを示す。

【0176】

【発明の効果】本発明によれば、伝送中にエラーが発生してもエラーが発生した部分のみ再伝送することによってネットワークの負担と伝送時間を縮め、伝送された一部の連結情報、幾何情報そして特性情報を用いて三次元メッシュを漸進的に復元しうる。

【図面の簡単な説明】

【図1】図1(A)乃至図1(F)は三角形メッシュの一例に対する頂点スパニンググラフと三角形スパニンググラフの生成方法を示す図面である。

【図2】図2(A)乃至図2(D)は位相幾何学的サーザリ技術の適用例を示す図面である。

【図3】ループを有する頂点スパニンググラフの一例を示す図面である。

【図4】位相幾何学的サーザリにおけるバウンディングループを構成する方法を示す図面である。

【図5】図5(A)及び図5(B)は各々多角形メッシュとそのデュアルグラフの例を示す図面である。

【図6】既存の三次元メッシュ情報の符号化方式を概念的に示す図面である。

【図7】階層的三次元メッシュ情報の表現を概念的に示す図面である。

【図8】図8(A)及び図8(B)は各々三次元メッシュ情報(MO)及び基本メッシュ情報(MOBL)を例示した図面である。

20 【図9】三角形ツリー/三角形データ対とバウンディンググループインデックスとの関係を概念的に示す図面である。

【図10】図10(A)乃至図10(C)は方向性情報を活用したコーディングを概念的に示す図面である。

【図11】図11(A)及び図11(B)は方向性情報の有無に応じる符号化順序を比較した図面である。

【図12】図12(A)乃至図12(I)は固定構造分割方法を示す図面である。

30 【図13】図13(A)及び図13(B)はデータ分割を概念的に示す図面である。

【図14】図14(A)乃至図14(D)は可変構造分割方法を示す図面である。

【図15】図15(A)乃至図15(D)は可変構造分割方法による分割を例示した図面である。

【図16】メインブランチにおけるデータ分割を例示した図面である。

【図17】三角形ツリーの平均ビット発生量を用いたデータ分割方法を示す図面である。

40 【図18】図18(A)乃至図18(C)は仮想連結成分の構成方法を示す図面である。

【図19】多角形メッシュの三角形メッシュ化を例示した図面である。

【図20】図20(A)乃至図20(E)は多角形メッシュにおける分割と関連文法構成を例示した図面である。

【図21】バウンディンググループのインデクシング方法とそれに係るパーティションのヘッダ情報をコーディングする方法を例示した図面である。

【図22】バウンディンググループを含むデータ分割を示す図面である。

50 【図23】図23(A)及び図23(B)はバウンディンググループ

(30)

特開 2000-194843

57

ブインデックス定義方法を例示した図面である。

【図 24】図 24(A)乃至図 24(E)は幾何情報の符号化方法を示す図面である。

【図 25】図 25(A)及び図 25(B)は幾何情報、色相情報、法線情報、テクスチャー座標情報に対する文法構成を示す図面である。

【図 26】幾何情報のコーディング方法を概念的に示す図面である。

【図 27】3次元メッシュに対する圧縮されたビットストリームの構成を示す図面である。

【図 28】3次元メッシュがエラー回復的なモードで符号化された場合の連結成分データブロックを示す図面である。

【図 29】図 28の連結成分データブロック内のレコードの構成を示す図面である。

【図 30】三角形データの整列を示す図面である。

【図 31】三角形データの構成を示す図面である。

【図 32】符号化文法の一例を示す図面である。

【図 33】符号化文法の一例を示す図面である。

【図 34】符号化文法の一例を示す図面である。

【図 35】符号化文法の一例を示す図面である。

【図 36】符号化文法の一例を示す図面である。

【図 37】符号化文法の一例を示す図面である。

【図 38】符号化文法の一例を示す図面である。

【図 39】符号化文法の一例を示す図面である。

【図 40】符号化文法の一例を示す図面である。

【図 41】符号化文法の一例を示す図面である。

【図 42】符号化文法の一例を示す図面である。

【図 43】符号化文法の一例を示す図面である。

10

20

\*

58

\* 【図 44】符号化文法の一例を示す図面である。

【図 45】符号化文法の一例を示す図面である。

【図 46】符号化文法の一例を示す図面である。

【図 47】符号化文法の一例を示す図面である。

【図 48】符号化文法の一例を示す図面である。

【図 49】符号化文法の一例を示す図面である。

【図 50】符号化文法の一例を示す図面である。

【図 51】符号化文法の一例を示す図面である。

【図 52】符号化文法の一例を示す図面である。

【図 53】符号化文法の一例を示す図面である。

【図 54】符号化文法の一例を示す図面である。

【図 55】符号化文法の一例を示す図面である。

【図 56】符号化文法の一例を示す図面である。

【図 57】符号化文法の一例を示す図面である。

【図 58】符号化文法の一例を示す図面である。

【図 59】符号化文法の一例を示す図面である。

【図 60】符号化文法の一例を示す図面である。

【図 61】符号化文法の一例を示す図面である。

【図 62】符号化文法の一例を示す図面である。

【図 63】符号化文法の一例を示す図面である。

【図 64】符号化文法の一例を示す図面である。

【図 65】符号化文法の一例を示す図面である。

【図 66】符号化文法の一例を示す図面である。

【図 67】符号化文法の一例を示す図面である。

【図 68】符号化文法の一例を示す図面である。

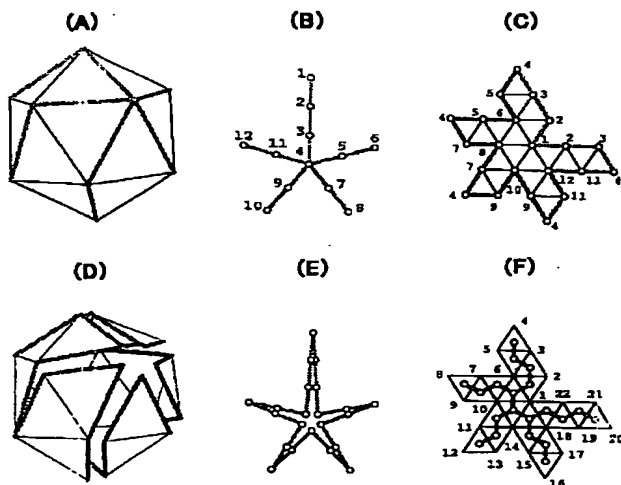
【図 69】符号化文法の一例を示す図面である。

【図 70】符号化文法の一例を示す図面である。

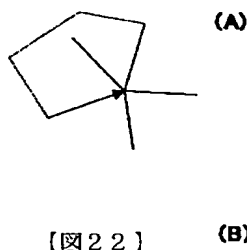
【図 71】符号化文法の一例を示す図面である。

【図 72】符号化文法の一例を示す図面である。

【図 1】

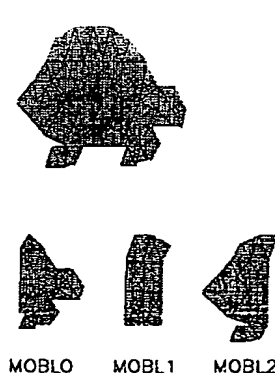


【図 3】



L	R	tt	td
---	---	----	----

【図 8】



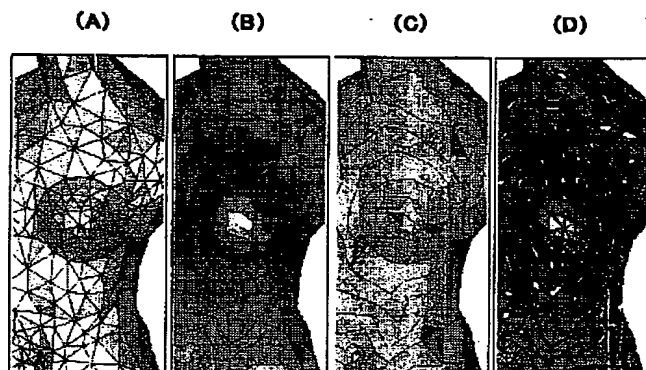
【図 27】

3D Mesh Header	CC Data #1	...	CC Data #nCC
----------------	------------	-----	--------------

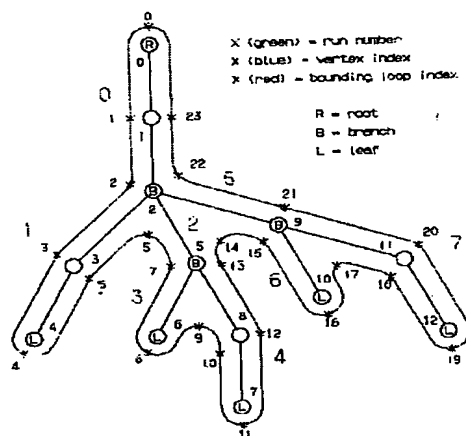
(31)

特開2000-194843

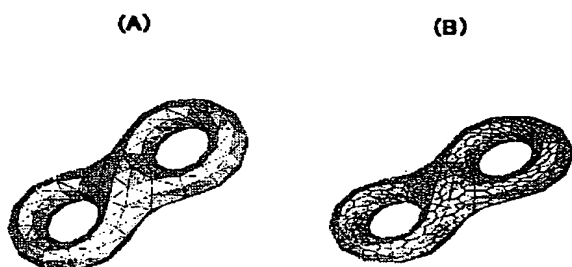
【図2】



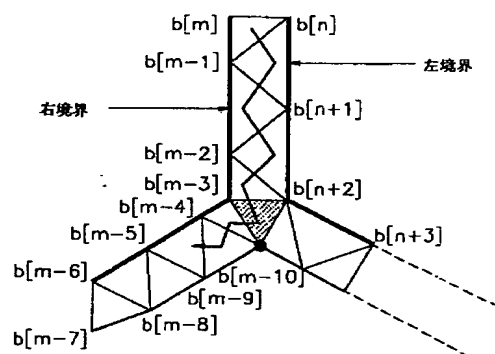
【図4】



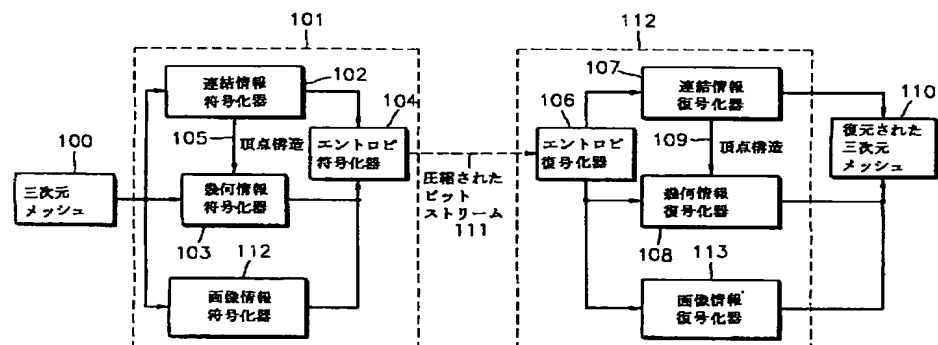
【図5】



【図9】



【図6】



【図28】

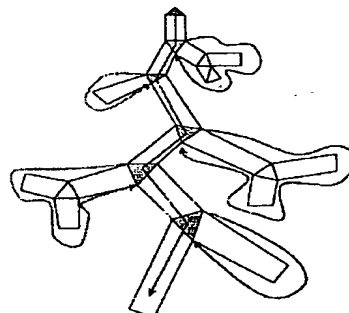
【図29】

パーティション1	パーティション2	...	パーティションn	PT	頂点グラフ	三角形ツリー	三角形データ
----------	----------	-----	----------	----	-------	--------	--------

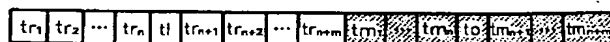


特開2000-194843

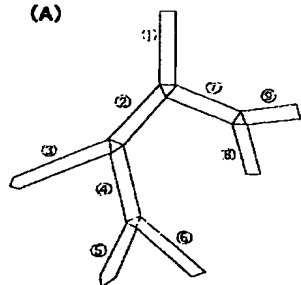
【圖 10】



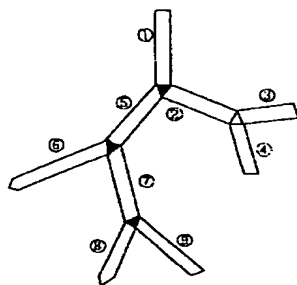
**(C)**



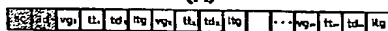
(A)



【圖 12】



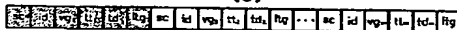
(A)



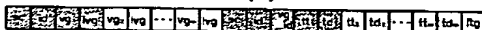
(B)



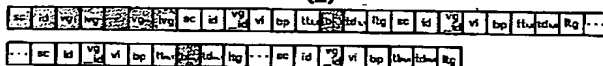
(c)



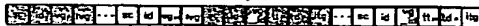
(D)



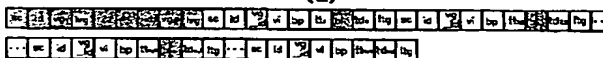
**(E)**



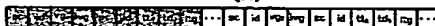
(F)



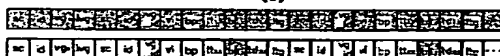
**(G)**



(H)



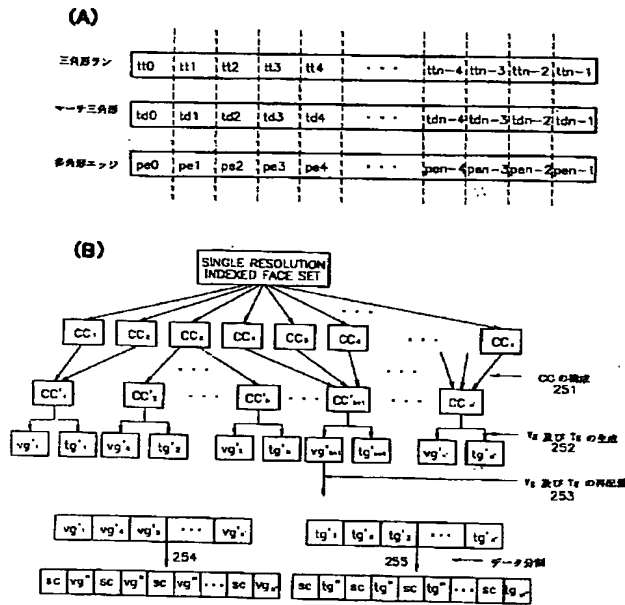
(1)



(33)

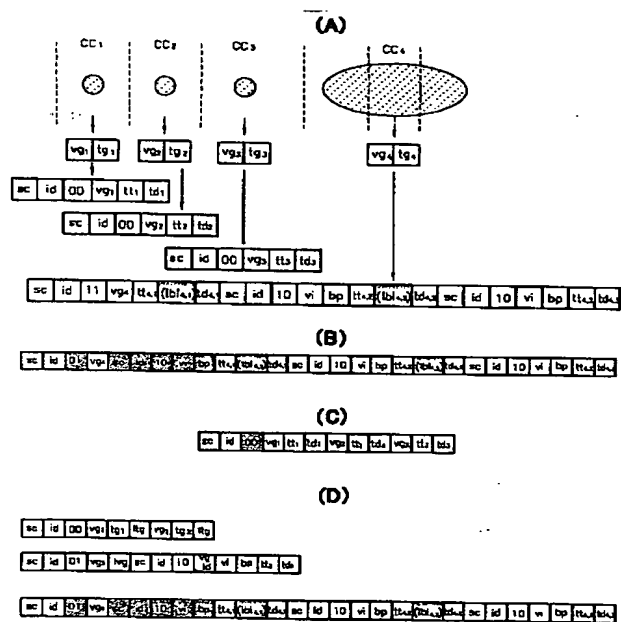
特開2000-194843

【図13】

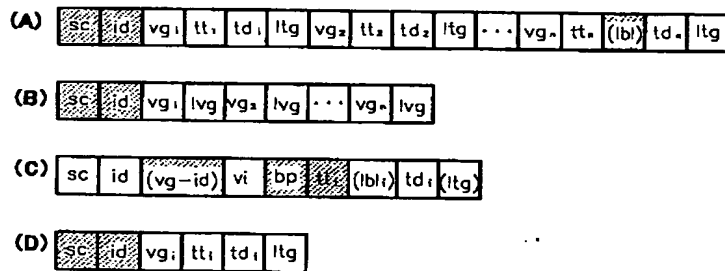


【図14】

【図15】



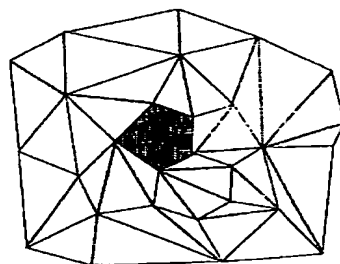
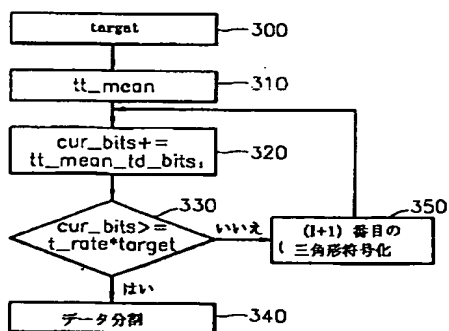
【図16】



【図17】

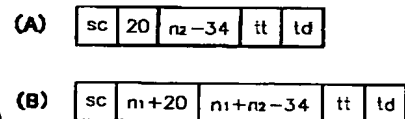
【図19】

【図23】



【図30】

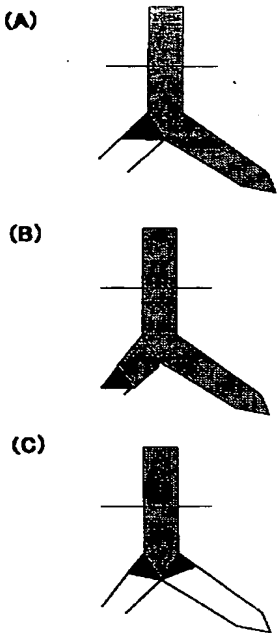
三角形#1に対するデータ	三角形#2に対するデータ	...	三角形#nTに対するデータ
--------------	--------------	-----	---------------



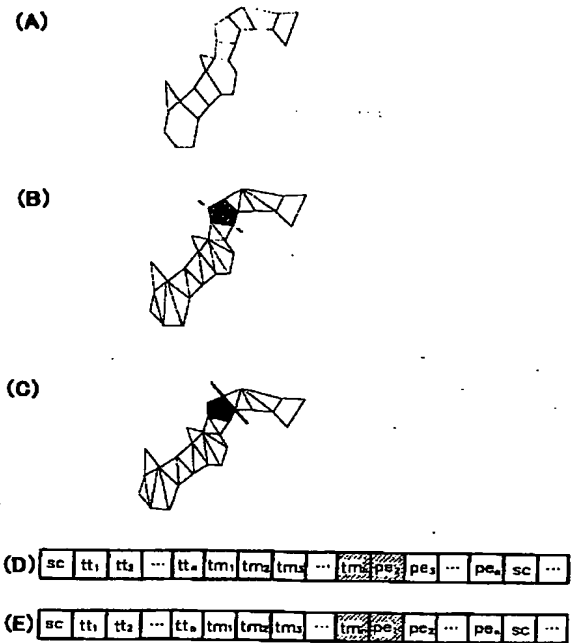
(34)

特開2000-194843

【図18】



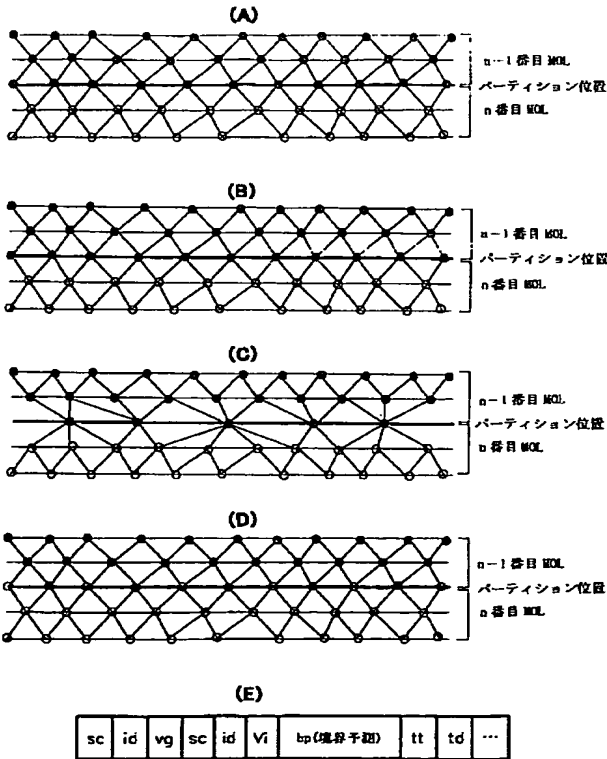
【図20】



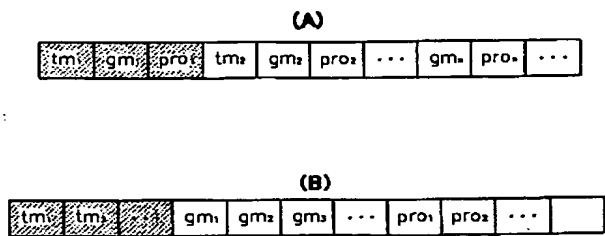
【図21】

bnd[0][0]	bnd[0]
bnd[0][1]	bnd[1]
bnd[0][2]	bnd[2]
...	...
bnd[0][n1-1]	bnd[n1-1]
bnd[1][0]	bnd[n1]
bnd[1][1]	bnd[n1+1]
...	...
bnd[0][n2-1]	bnd[n1+n2-1]
bnd[2][0]	bnd[n1+n2]
bnd[2][1]	bnd[n1+n2+1]
bnd[2][2]	bnd[n1+n2+2]
...	...
bnd[0][n3-1]	bnd[n1+n2+n3-1]
...	...
bnd[m-1][nm-1]	bnd[n1+n2+n3+...+nm-1]

【図24】



【図25】



【図31】

marching pattern	td_orientation	polygon_edge	coord	Normal	color	texCoord
---------------------	----------------	--------------	-------	--------	-------	----------

【図44】

1.2.1.4 triangle_data		
triangle_data {		
qf_decoded(triangulated, triangulated_coord)	No. of bits	variable
depth=0		
root_triangle {		
for (i=1; i<=nbtriangle; i++)		
triangle {		
}		
}		

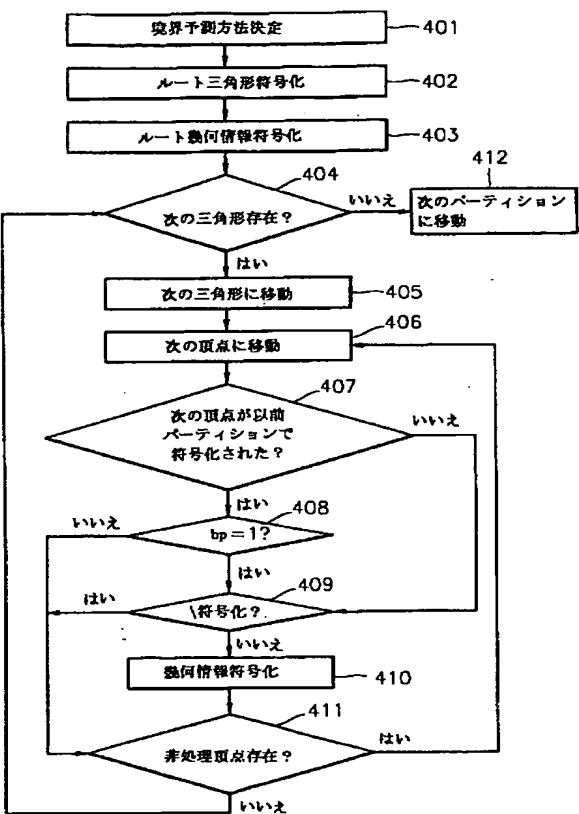
【図33】

1.2.1.3 3D_Mesh_Object_Layer		
3D_Mesh_Object_Layer {		
3D_Mesh_start_code	18	variable
mol_id	8	variable
if (mol_id == 1) {		
cpl_n_vertices	24	variable
cpl_n_triangles	24	variable
cpl_n_edges	24	variable
}		
if (mol_id == 0)		
3D_Mesh_Object_Base_Layer {		
}		
Else		
3D_Mesh_Object_Extremes_Layer {		
}		

【図55】

mol_coord_sample {		
for (i=0; i<3; i++)		
for (j=0; j<=coord_point; j++)		
qf_decoded(coord_id, coord_coord)		variable
}		

【図26】



【図35】

1.2.1.5 coord_header		
coord_header {		
coord_headering	2	variable
coord_nbox	1	variable
if (coord_nbox == 1) {		
coord_xmin	32	variable
coord_xmax	32	variable
coord_ymin	32	variable
coord_ymax	32	variable
}		
coord_point	8	variable
coord_pred_type	2	variable
if (coord_pred_type == "tree_prediction") {		
coord_pred_type == "parallel_prediction" {		
for (i=1; i<=coord_ntriangle; i++)		
coord_ntriangle	4-32	variable
}		
}		

(36)

特開 2000-194843

【図 32】

## 1.2 Visual bitstream syntax

## 1.2.1 3D Mesh Object

## 1.2.1.1 3D\_Mesh\_Object

3D_Mesh_Object ()	No. of bits	Mnemonic
3D_MESH_start_code	16	uint16
3D_Mesh_Object_Header()		
do {		
3D_Mesh_Object_Layer()		
} while (nextbits_tested() == 3D_MESH_start_code)		
}		

## 1.2.1.2 3D\_Mesh\_Object\_Header

3D_Mesh_Object_Header ()	No. of bits	Mnemonic
Ccw	1	bool
Convex	1	bool
Solid	1	bool
CrashesAngle	0	uint16
coord_header()		
normal_header()		
color_header()		
texCoord_header()		
cpl_data	1	bool
if (cpl_data == 1)		
cpl_header()		
}		

【図 36】

## 1.2.1.6 normal\_header

normal_header ()	No. of bits	Mnemonic
normal_binning	2	uint16
if (normal_binning != "not_bound") {		
normal_bbox	1	bool
normal_quant	8	uint16
normal_pred_type	2	uint16
if ( normal_pred_type=="true_prediction")		
normal_pred_type=="parallelogram_prediction") {		
normal_planebits	2	uint16
for (i=1; i<normal_planebits; i++)		
normal_planebits	4-32	uint16
}		
}		
}		

【図 40】

## 1.2.1.10 connected\_component

connected_component ()	No. of bits	Mnemonic
vector_graph()		
qf_decode(hes_sitiches, hes_sitiches_count)		uint16
if (hes_sitiches == 1)		
sitiches()		
triangle_tree()		
triangle_data()		
}		

【図 34】

## 1.2.1.4 3D\_Mesh\_Object\_Base\_Layer

3D_Mesh_Object_Base_Layer	No. of bits	Mnemonic
do {		
3D_MESH_start_code	16	uint16
mesh_id	8	uint16
while (typesigned())		
mesh_id	1	bool
qf_decode()		
if (3D_MESH_start_code == "partition_type_0") {		
do {		
connected_component()		
qf_decode(hes_sitiches, hes_sitiches_count)		uint16
while (hes_sitiches == 1)		
}		
else if (3D_MESH_start_code == "partition_type_1") {		
vq_number=0		
do {		
vector_graph()		
vq_number++		
qf_decode(coding_left_vq, coding_left_vq_count)		uint16
while (coding_left_vq == 1)		
}		
else if (3D_MESH_start_code == "partition_type_2") {		
cpl_number > 1		
qf_decode(coding_vq_id, coding_vq_id_count)		uint16
qf_decode( coding_left_block_id,		uint16
coding_left_block_id_count)		
qf_decode( coding_right_block_id,		uint16
coding_right_block_id_count)		
qf_decode( coding_hes_pred,		uint16
coding_hes_pred_count)		
triangle_tree()		
triangle_data()		
}		
while (nextbits_tested() == 3D_MESH_start_code)		
}		

【図 37】

## 1.2.1.7 color\_header

color_header ()	No. of bits	Mnemonic
color_binning	2	uint16
if (color_binning != "not_bound") {		
color_bbox	1	bool
if (color_bbox == 1) {		
color_rmin	32	bool
color_rmax	32	bool
color_gmin	32	bool
color_gmax	32	bool
color_bmin	32	bool
color_bmax	32	bool
}		
color_quant	5	uint16
color_pred_type	2	uint16
if ( color_pred_type=="true_prediction")		
color_pred_type=="parallelogram_prediction") {		
color_planebits	2	uint16
for (i=1; i<color_planebits; i++)		
color_planebits	4-32	uint16
}		
}		
}		



(38)

特開2000-194843

【図43】

## 1.3.1.13 triangle\_tree

	No. of bits	Mnemonic
triangle_tree {		
depth = 0		
ntriangles = 0		
branch_position = 0		
do {		
q_decode(tl_run_length, tl_run_length_context)		visit
ntriangles += tl_run_length		
q_decode(tl_leaf, tl_leaf_context)		visit
if (tl_leaf == 1) {		
depth--		
}		
else {		
branch_position = ntriangles		
depth++		
}		
} while (depth >= 0)		
if (tl_MOBI_start_code == "partition_type_2")		
if (codeop_right_loop_idx - codeop_left_loop_idx - 1 > ntriangles) {		
if (branch_position == ntriangles - 2) {		
q_decode(codeop_branch_len, tl)		visit
codeop_branch_len_context		
ntriangles = 2		
}		
}		
else		
ntriangles--		
}		

【図45】

## 1.3.1.18 root\_triangle

	No. of bits	Mnemonic
root_triangle {		
if (searching_triangle)		
q_decode(searching_pattern, searching_pattern_context(searching_pattern))		visit
else {		
if (tl_MOBI_start_code == "partition_type_2")		
if (tl_leaf == 0 && depth == 0)		
q_decode(tl_orbitalsize, tl_orbitalsize_context)		visit
if (tl_leaf == 0)		
depth++		
else		
depth--		
}		
if (tl_MOBI_start_code == "partition_type_2")		
if (tl_orbitalsize == 0)		
q_decode(polygon_edges, polygon_edges_context(polygon_edges))		visit
}		
root_coord()		
root_normal()		
root_color()		
root_line_coord()		
}		

【図46】

	No. of bits	Mnemonic
root_coord {		
if (tl_MOBI_start_code == "partition_type_2") {		
if (visited(vertex_index) == 0) {		
root_coord_sample()		
}		
if (visited(vertex_index) == 0) {		
coord_sample()		
coord_sample()		
}		
}		
else {		
root_coord_sample()		
coord_sample()		
coord_sample()		
}		

【図48】

	No. of bits	Mnemonic
root_color {		
if (color_binding != "not_bound")		
if (tl_MOBI_start_code == "partition_type_2") {		
if (color_binding != "bound_per_vector") {		
visited(vertex_index) == 0 {		
root_color_sample()		
}		
if (color_binding != "bound_per_face" && (color_binding != "bound_per_vector")) {		
visited(vertex_index) == 0 {		
color_sample()		
color_sample()		
}		
}		
else {		
root_color_sample()		
}		
if (color_binding != "bound_per_face") {		
color_sample()		
color_sample()		
}		

【図51】

	No. of bits	Mnemonic
coord {		
if (tl_MOBI_start_code == "partition_type_2") {		
if (visited(vertex_index) == 0)		
if (no_samples)		
root_coord_sample()		
else		
coord_sample()		
}		
}		
else {		
if (visited(vertex_index) == 0)		
coord_sample()		
}		

(39)

特開2000-194843

【図47】

root_normal()	No. of bits	Mnemonic
if (normal_binding != "not_bound")		
if (MO_MOBI_start_code == "partition_type_2") {		
if (normal_binding != "bound_per_vector") {		
visited[vertex_index]=0;		
root_normal_sample()		
if (normal_binding != "bound_per_face" && normal_binding != "bound_per_vector"    visited[vertex_index]==0){		
normal_sample()		
normal_sample()		
}		
}		
else {		
root_normal_sample()		
if (normal_binding != "bound_per_face") {		
normal_sample()		
normal_sample()		
}		
}		

【図49】

root_textCoord()	No. of bits	Mnemonic
if (textCoord_binding != "not_bound")		
if (MO_MOBI_start_code == "partition_type_2") {		
if (textCoord_binding != "bound_per_vector") {		
visited[vertex_index]=0;		
root_textCoord_sample()		
if (textCoord_binding != "bound_per_vector"    visited[vertex_index]==0){		
textCoord_sample()		
textCoord_sample()		
}		
}		
else {		
root_textCoord_sample()		
textCoord_sample()		
textCoord_sample()		
}		
}		

【図50】

5.2.1.18 Interpale

triangle()	No. of bits	Mnemonic
if (marching_triangle)		
q_decode(marching_pattern, marching_pattern_codes(marching_pattern))		vector
else {		
if (MO_MOBI_start_code == "partition_type_2")		
if (is_leaf == "Y" && depth==0)		
q_decode(p1_orientation, b1_orientation_codes)		vector
if (is_leaf == "Y")		
depth++		
else		
depth--		
if (is_triangle == "Y")		
q_decode(polygons_edges, polygon_edges_codes(polygon_edges))		vector
coord()		
normal()		
color()		
textCoord()		
}		

【図52】

normal()	No. of bits	Mnemonic
if (normal_binding == "bound_per_vector") {		
if (MO_MOBI_start_code == "partition_type_2") {		
if (visited[vertex_index]==0)		
if (no_vectors)		
root_normal_sample()		
else		
normal_sample()		
}		
else {		
if (visited[vertex_index]==0)		
normal_sample()		
}		
} else if (normal_binding == "bound_per_face") {		
if (is_triangle == "Y"    polygon_edges == "Y")		
normal_sample()		
} else if (normal_binding == "bound_per_vector") {		
if (is_triangle == "Y"    polygon_edges == "Y") {		
normal_sample()		
normal_sample()		
}		
normal_sample()		
}		



(40)

特開 2000-194843

【図 53】

color()	No. of bits	Mnemonics
if (color_binning == "bound_per_vector") {		
if (AD_MODEL_start_code == "partition_type_2") {		
if (visited[vertex_index] == 0)		
if (no_smc_start)		
root_color_sample()		
else		
color_sample()		
}		
} else {		
if (visited[vertex_index] == 0)		
color_sample()		
}		
} else if (color_binning == "bound_per_corner") {		
if (triangulated == "1"    polygon_edge == "1")		
color_sample()		
} else if (color_binning == "bound_per_corner") {		
if (triangulated == "1"    polygon_edge == "1") {		
color_sample()		
color_sample()		
}		
color_sample()		
}		

【図 54】

texCoord()	No. of bits	Mnemonics
if (texCoord_binning == "bound_per_vector") {		
if (AD_MODEL_start_code == "partition_type_2") {		
if (visited[vertex_index] == 0)		
if (no_smc_start)		
root_texCoord_sample()		
else		
texCoord_sample()		
}		
} else {		
if (visited[vertex_index] == 0)		
texCoord_sample()		
}		
} else if (texCoord_binning == "bound_per_corner") {		
if (triangulated == "1"    polygon_edge == "1") {		
texCoord_sample()		
texCoord_sample()		
}		
texCoord_sample()		
}		

【図 57】

【図 56】

root_normal_sample()	No. of bits	Mnemonics
for (p=0; p<3; p++)		
for (q=0; q<normal_quant; q++)		
qf_decode(normal_bit, zero_context)		vector
}		
root_color_sample()	No. of bits	Mnemonics
for (p=0; p<3; p++)		
for (q=0; q<color_quant; q++)		
qf_decode(color_bit, zero_context)		vector
}		
root_texCoord_sample()	No. of bits	Mnemonics
for (p=0; p<2; p++)		
for (q=0; q<texCoord_quant; q++)		
qf_decode(texCoord_bit, zero_context)		vector
}		

coord_sample()	No. of bits	Mnemonics
for (p=0; p<3; p++) {		
p=0		
do {		
qf_decode(coord_leading_bit,		vector
coord_leading_bit_context(p+1))		
p++		
} while ((coord_quant && coord_leading_bit == 0))		
if (coord_leading_bit == "1") {		
qf_decode(coord_sign_bit, zero_context)		vector
} do {		
qf_decode(coord_trailing_bit,		vector
zero_context)		
} while ((coord_quant))		
}		

【図 62】

## 1.2.1.17.1 pre\_smoothing parameters

pre_smoothing_parameters()	No. of bits	Mnemonics
pre_smoothing_s	8	short
pre_smoothing_lambda	32	float
pre_smoothing_rho	32	float
}		

## 1.2.1.17.2 post\_smoothing parameters

post_smoothing_parameters()	No. of bits	Mnemonics
post_smoothing_s	8	short
post_smoothing_lambda	32	float
post_smoothing_rho	32	float
}		

(41)

特開 2000-194843

【図 58】

normal_sample0 {	No. of bits	Mnemonic
for (p=0; k<1; k++) {		
j=0		
do {		
qf_decode(normal_leading_bit,		
normal_leading_bit_context[j])		vector
j++		
} while (j<normal_quant && normal_leading_bit == 0)		
if (normal_leading_bit == 1) {		
qf_decode(normal_sign_bit, zero_context)		vector
do {		
qf_decode(normal_trailing_bit,		
zero_context)		vector
} while (j<normal_quant)		
}		
}		

【図 59】

color_sample0 {	No. of bits	Mnemonic
for (p=0; k<2; k++) {		
j=0		
do {		
qf_decode(color_leading_bit,		
color_leading_bit_context[2*j+k])		vector
j++		
} while (j<color_quant && color_leading_bit == 0)		
if (color_leading_bit == 1) {		
qf_decode(color_sign_bit, zero_context)		vector
do {		
qf_decode(color_trailing_bit,		
zero_context)		vector
} while (j<color_quant)		
}		
}		

【図 60】

texCoord_sample0 {	No. of bits	Mnemonic
for (p=0; k<2; k++) {		
j=0		
do {		
qf_decode(texCoord_leading_bit,		
texCoord_leading_bit_context[2*j+k])		vector
j++		
} while (j<texCoord_quant && texCoord_leading_bit == 0)		
if (texCoord_leading_bit == 1) {		
qf_decode(texCoord_sign_bit, zero_context)		vector
do {		
qf_decode(texCoord_trailing_bit,		
zero_context)		vector
} while (j<texCoord_quant)		
}		
}		

【図 61】

1.2.1.17 SOMeshObjRefinement\_Layer

SOMeshObjRefinement_Layer 0 {	No. of bits	Mnemonic
do {		
3D_MESH_start_code	16	scalar
mesh_id	8	scalar
connectivity_update	2	scalar
pre_smoothing	1	bool
if (pre_smoothing == 1) {		
pre_smoothing_parameters()		
post_smoothing	1	bool
if (post_smoothing == 1) {		
post_smoothing_parameters()		
while (bytestring)		
one_bit	1	bool
q_start()		
if (connectivity_update == "a_update")		
a_pre_update()		
if (pre_smoothing == 1)    post_smoothing == 1		
smoothing_constants()		
/* apply pre smoothing step 1 */		
if (connectivity_update == "a_update")		
a_post_update()		
if (connectivity_update != "a_updated")		
qf_decode(other_update, zero_context)		vector
if (connectivity_update == "a_updated"		
other_update == 1)		
other_property_update()		
/* apply post smoothing step 1 */		
} while (bytestring_bytestring) == 3D_MESH_start_code		
}		

【図 68】

loop_texCoord_update 0 {	No. of bits	Mnemonic
if (texCoord_binning == "board_per_vertex") {		
for each tree loop vertex		
texCoord_sample0		
}		
else if (texCoord_binning == "board_per_corner") {		
for each tree loop corner		
texCoord_sample0		
}		
}		

(42)

特開 2000-194843

【図 63】

## 1.2.1.17.3 ts\_pre\_update

	No. of bits	Mnemonic
ts_pre_update() {		
for each connected component {		
forest()		
for each tree in forest {		
triangle_tree()		
/* for each tree loop vertex not visited[vector_index]*1^V		
triangle_data()		
}		
}		
}		

	No. of bits	Mnemonic
forest() {		
for each edge in connected component		
if (forest no loop in forest)		
cf_decode(ts_pre_update_forest_edges,		vector
pts_forest_edges_content)		
}		

【図 65】

## 1.2.1.17.5 ts\_post\_update

	No. of bits	Mnemonic
ts_post_update() {		
for each connected component {		
for each tree in forest		
tree_loop_property_update()		
}		

	No. of bits	Mnemonic
tree_loop_property_update() {		
loop_coord_update()		
loop_normal_update()		
loop_color_update()		
loop_texture_update()		
}		

	No. of bits	Mnemonic
loop_coord_update() {		
for each tree loop vertex		
coord_sample()		
}		

【図 69】

## 1.2.1.17.6 other\_property\_update

	No. of bits	Mnemonic
other_property_update() {		
other_coord_update()		
other_normal_update()		
other_color_update()		
other_texture_update()		
}		

	No. of bits	Mnemonic
other_coord_update() {		
for each vertex in mesh		
if (vertex is not a tree loop vertex)		
coord_sample()		
}		

【図 64】

## 1.2.1.17.4 smoothing\_constants

	No. of bits	Mnemonic
smoothing_constants() {		
cf_decode(smooth_with_sharp_edges, zero_coord)		vector
if (smooth_with_sharp_edges == '1')		
sharp_edges_marks()		
cf_decode(smooth_with_bound_vertices, zero_coord)		vector
if (smooth_with_bound_vertices == '1')		
bound_vertices_marks()		
}		

	No. of bits	Mnemonic
sharp_edges_marks() {		
for each edge		
cf_decode(smooth_sharp_edges,		vector
smooth_sharp_edges_content)		
}		

	No. of bits	Mnemonic
bound_vertices_marks() {		
for each vertex		
cf_decode(smooth_bound_vertices,		vector
smooth_bound_vertices_content)		
}		

【図 66】

	No. of bits	Mnemonic
loop_normal_update() {		
if (normal_binding == "bound_per_vector") {		
for each tree loop vertex		
normal_sample()		
}		
else if (normal_binding == "bound_per_face") {		
for each tree loop face		
normal_sample()		
}		
else if (normal_binding == "bound_per_corner") {		
for each tree loop corner		
normal_sample()		
}		
}		

【図 67】

	No. of bits	Mnemonic
loop_color_update() {		
if (color_binding == "bound_per_vector") {		
for each tree loop vertex		
color_sample()		
}		
else if (color_binding == "bound_per_face") {		
for each tree loop face		
color_sample()		
}		
else if (color_binding == "bound_per_corner") {		
for each tree loop corner		
color_sample()		
}		
}		

(43)

特開2000-194843

【図70】

other_normal_update () {	No. of bits	Mnemonic
if (normal_binding == "bound_per_vertex") {		
for each vertex in mesh		
if (vertex is not a true loop vertex)		
normal_sample()		
}		
else if (normal_binding == "bound_per_face") {		
for each face in mesh		
if (face is not a true loop face)		
normal_sample()		
}		
else if (normal_binding == "bound_per_corner") {		
for each corner in mesh		
if (corner is not a true loop corner)		
normal_sample()		
}		
}		

【図71】

other_color_update () {	No. of bits	Mnemonic
if (color_binding == "bound_per_vertex") {		
for each vertex in mesh		
if (vertex is not a true loop vertex)		
color_sample()		
}		
else if (color_binding == "bound_per_face") {		
for each face in mesh		
if (face is not a true loop face)		
color_sample()		
}		
else if (color_binding == "bound_per_corner") {		
for each corner in mesh		
if (corner is not a true loop corner)		
color_sample()		
}		
}		

【図72】

other_texCoord_update () {	No. of bits	Mnemonic
if (texCoord_binding == "bound_per_vertex") {		
for each vertex in true loop		
if (vertex is not a true loop vertex)		
texCoord_sample()		
}		
else if (texCoord_binding == "bound_per_corner") {		
for each corner in mesh		
if (corner is not a true loop corner)		
texCoord_sample()		
}		
}		

フロントページの続き

(72)発明者 張 義善  
大韓民国京畿道水原市八達区靈通洞シンナムシル963- 2 番地 双龍アパート542棟904号

(72)発明者 韓 万鎮  
大韓民国ソウル特別市瑞草区瑞草洞1619- 6 番地

(72)発明者 鄭 錫潤  
大韓民国ソウル特別市瑞草区蚕院洞52- 2 番地 新盤浦13次アパート328棟601号

(72)発明者 徐 亮錫  
大韓民国ソウル特別市松坡区風納洞219番地 美星アパート3 棟501号